The era of Industry 4.0 opens up the possibility of optimizing production systems in a data-driven way. To turn data into value, machine learning (ML) models are trained on production data aiming at identifying patterns to optimize processes. A crucial prerequisite for achieving performant ML models is the availability of high quality data. Since raw data generated in production exhibits multiple quality issues, data preprocessing (DPP) is required to increase the data quality. One of the key design decisions in any ML project is the choice of suitable DPP methods. The search space further increases when DPP methods are configured into DPP pipelines. Due to the high number of possible DPP pipelines, data scientists commonly select suitable pipelines manually via trial and error. For these reasons, DPP nowadays accounts for approximately 80 % of the time in ML projects.

To guide data scientists, decision support systems (DSS) have been developed that assist in the selection of suitable DPP pipelines but do not cover production-specific requirements. Therefore, the main research question was: Can a DSS be developed that supports in recommending DPP pipelines for ML applications in production?

To be able to answer the main research question, a meta learning-based decision support system, called Meta-DPP, was developed. Meta-DPP relies on three core components: the meta target selector, meta features database, and meta model. The *meta target selector* chooses between two preselected sets of overall well performing pipelines, called pipeline pools, for both classification and regression tasks. Further, the *meta features database* stores learning task-specific information about the data set, e. g., the number of instances. The *meta model* then recommends a pipeline from the pipeline pool based on the meta features from the database. When applying Meta-DPP, a user interface enables the data scientist, or production expert to input their data set, learning task, ML algorithm and information about explainability. Given these four inputs, Meta-DPP provides a ranked recommendation of the DPP pipelines from the pool.

Verifying and validating revealed the correct development and implementation of Meta-DPP. The validation on 324 production use cases further prove that Meta-DPP outperform essential pipelines on average, whereby essential pipelines ensure the functioning of ML algorithms by performing minimum DPP. Thus, the main research question was positively answered.

ISBN 978-3-98555-147-7

PRODUKTIONSQUALITÄT UND MESSTECHNIK

PRODUKTIONSQUALITÄT UND MESSTECHNIK

Recommending Data Preprocessing Pipelines for Machine Learning Applications in Production

Maik Frye

ERGEBNISSE AUS DER PRODUKTIONSTECHNIK

Maik Frye

Recommending Data Preprocessing Pipelines for Machine Learning Applications in Production

WZL | RWTH AACHEN UNIVERSITY

Fraunhofer IPT

Edition Wissenschaft Apprimus

Recommending Data Preprocessing Pipelines for Machine Learning Applications in Production

Empfehlung von Pipelines der Datenvorverarbeitung für Anwendungen des maschinellen Lernens in der Produktion

Von der Fakultät für Maschinenwesen
der Rheinisch-Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften
genehmigte Dissertation

vorgelegt von

Maik Frye

Berichter/in:

Univ.-Prof. Dr.-Ing. Robert Heinrich Schmitt
Univ.-Prof. Marek Behr, Ph. D.

Tag der mündlichen Prüfung: 01. September 2022

ERGEBNISSE AUS DER PRODUKTIONSTECHNIK

Maik Frye

Recommending Data Preprocessing Pipelines for Machine Learning Applications in Production

Herausgeber:
Prof. Dr.-Ing. T. Bergs
Prof. Dr.-Ing. Dipl.-Wirt. Ing. G. Schuh
Prof. Dr.-Ing. C. Brecher
Prof. Dr.-Ing. R. H. Schmitt

Band 3/2023

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über https://portal.dnb.de abrufbar.

Maik Frye:

Recommending Data Preprocessing Pipelines for Machine Learning Applications in Production

1. Auflage, 2023

Gedruckt auf holz- und säurefreiem Papier, 100% chlorfrei gebleicht.

Apprimus Verlag, Aachen, 2023
Wissenschaftsverlag des Instituts für Industriekommunikation und Fachmedien
an der RWTH Aachen
Steinbachstr. 25, 52074 Aachen
Internet: www.apprimus-verlag.de, E-Mail: info@apprimus-verlag.de

Printed in Germany

ISBN 978-3-98555-147-7

D 82 (Diss. RWTH Aachen University, 2022)

Vorwort

Diese Arbeit entstand im Rahmen meiner Tätigkeit als wissenschaftlicher Mitarbeiter in der Abteilung Produktionsqualität am Fraunhofer-Institut für Produktionstechnologie IPT in Aachen. Ich danke meinem Doktorvater Prof. Robert Heinrich Schmitt für die Betreuung meiner Dissertation. Außerdem danke ich Prof. Marek Behr als zweiten Berichter sowie Prof. Flemisch als Vorsitz der Promotionskommission. Ich werde sehr positiv auf den 01. September 2022, den Tag meiner Promotion, zurückblicken.

Einen großen Dank möchte ich an alle Kolleginnen und Kollegen aussprechen, die ich in den letzten Jahren am Fraunhofer IPT und anderen Instituten kennenlernen durfte. Ich empfand es immer als sehr großes Privileg mit so motivierten Menschen arbeiten und diese gleichzeitig meine Freunde nennen zu dürfen.

Im Besonderen möchte ich allen danken, die zum Erfolg dieser Arbeit maßgeblich beigetragen haben. Hierbei danke ich Dr. Thomas Vollmer für die große Unterstützung im Erstellungs- und Abgabeprozess der Dissertation. Gerne hebe ich Dr. Jonathan Krauß hervor, der in der Themenfindung sowie Ausarbeitung der Dissertation immer zur Seite stand. Gleiches gilt für Dennis Grunert, der insbesondere auf den letzten Metern mir alles an Arbeit abgenommen hat, damit ich diesen Titel in den Händen halten darf. Ich danke für das immer ehrliche Feedback und der seelischen Unterstützung auch Hendrik Mende, Lars Leyendecker, Dr. Raphael Kiesel, Karl Lossie und Dr. Lars Nolting.

Außerdem haben an dieser Arbeit studentische Mitarbeitende tatkräftig mitgewirkt. Ich bin Thuany Karoline Stuart, Lukas Alberto Belck, Charlie Albietz, Johannes Beneke und David Steiner Sand zutiefst dankbar, dass sie so viel Leidenschaft in diese Arbeit gesteckt haben wie ich. Ein großer Dank gilt außerdem Sarah Grewe, Bruno Machado Pacheco sowie Petros Tsialis und Johannes Mohren.

Ich möchte meinen Freundinnen und Freunden aus Aachen und der Heimat Barßel danken für all die schönen Momente abseits der Dissertation.

Abschließend möchte ich mich bei meiner Familie und meiner lieben Frau Mandy bedanken. Danke für all die Unterstützung während meines Studiums, meiner Arbeit und der Dissertation. Ich werde meinen Eltern, meiner Schwester und meiner Frau Mandy ewig dankbar sein, dass sie in dieser Zeit so viel Rücksicht genommen haben und mich darüber hinaus in jeder Situation unterstützen. Ihnen widme ich diese Dissertation.

Aachen, 22. Januar 2023

Maik Frye

Zusammenfassung

Das Zeitalter der Industrie 4.0 ermöglicht die datengetriebene Optimierung von Produktionssystemen. Um einen Mehrwert aus Produktionsdaten zu generieren, werden Modelle des maschinellen Lernens (ML) eingesetzt. Eine entscheidende Voraussetzung für leistungsfähige ML-Modelle ist die Verfügbarkeit von Daten in hoher Qualität. Da die in der Produktion erzeugten Rohdaten verschiedenste Qualitätsmängel aufweisen, ist eine zielgerichtete Datenvorverarbeitung (DPP) erforderlich. Eine der wichtigsten Designentscheidungen in ML-Projekten ist die Wahl geeigneter DPP-Methoden. Der Suchraum vergrößert sich weiter, wenn mehrere DPP-Methoden in DPP-Pipelines konfiguriert werden. Aufgrund der großen Anzahl möglicher DPP-Pipelines wählen Data Scientists Pipelines in der Regel manuell und durch ein Trial and Error Verfahren aus. Daher nimmt DPP heutzutage etwa 80 % der Zeit in ML-Projekten in Anspruch. Um Data Scientists zu unterstützen, wurden Entscheidungsunterstützungssysteme entwickelt, die bei der Auswahl geeigneter DPP-Pipelines helfen, aber keine produktionsspezifischen Anforderungen abdecken. Daraus ergab sich die Hauptforschungsfrage der vorliegenden Dissertation: Kann ein Entscheidungsunterstützungssystem entwickelt werden, das bei der Empfehlung von DPP-Pipelines für ML-Anwendungen in der Produktion unterstützt?

Um die Hauptforschungsfrage zu beantworten, wurde ein Meta-Learning-basiertes Entscheidungsunterstützungssystem, Meta-DPP genannt, entwickelt. Meta-DPP stützt sich auf drei Kernkomponenten: den *Meta Target Selector*, die *Meta Features Database* und das *Meta Modell*. Der *Meta Target Selector* wählt zwischen zwei vorselektierten Mengen von performanten Pipelines, sog. Pipeline Pools, für Klassifizierungs- und Regressionsaufgaben aus. Darüber hinaus speichert die *Meta Features Database* lernaufgabenspezifische Informationen über den Datensatz, z. B. die Anzahl der Instanzen, sowie Performanzen von ML-Algorithmen und DPP-Pipelines. Das *Meta Modell* empfiehlt dann eine Pipeline aus dem Pipeline-Pool auf der Grundlage der Meta Features aus der Database. Bei der Anwendung von Meta-DPP kann der Data Scientist, Produktionsexperte oder IT-Experte über eine Benutzeroberfläche seinen Datensatz, Lernaufgabe, ML-Algorithmus und Informationen zur Erklärbarkeit eingeben. Auf Basis dieser vier Eingaben liefert Meta-DPP eine Rangfolge von Empfehlungen für die DPP-Pipelines aus dem Pool. Die Verifizierung und Validierung zeigte die korrekte Entwicklung und Implementierung von Meta-DPP. Die Validierung an 324 produktiven Anwendungsfällen zeigt außerdem, dass Meta-DPP im Durchschnitt besser abschneidet als essentielle Pipelines, wobei essentielle Pipelines das Funktionieren von ML-Algorithmen durch minimale DPP sicherstellen. Daher wurde die Hauptforschungsfrage positiv beantwortet.

Summary

The era of Industry 4.0 opens up the possibility of optimizing production systems in a data-driven way. To turn data into value, machine learning (ML) models are trained on production data aiming at identifying patterns to optimize processes. A crucial prerequisite for achieving performant ML models is the availability of high quality data. Since raw data generated in production exhibits multiple quality issues, data preprocessing (DPP) is required to increase the quality of the data. One of the key design decisions in any ML project is the choice of suitable DPP methods. The search space further increases when DPP methods are configured into DPP pipelines. Due to the high number of possible DPP pipelines, data scientists commonly select suitable pipelines manually and via trial and error. For these reasons, DPP nowadays accounts for approximately 80 % of the time in ML projects.

To guide data scientists, decision support systems (DSS) have been developed that assist in the selection of suitable DPP pipelines but do not cover production-specific requirements. Therefore, the main research question was: Can a DSS be developed that supports in recommending DPP pipelines for ML applications in production?

To be able to answer the main research question, a meta learning-based decision support system, called Meta-DPP, was developed. Meta-DPP relies on three core components: the meta target selector, meta features database, and meta model. The *meta target selector* chooses between two preselected sets of overall well performing pipelines, called pipeline pools, for both classification and regression tasks. Further, the *meta features database* stores learning task-specific information about the data set, e. g., the number of instances, as well as past ML algorithm and DPP pipeline performances. The *meta model* then recommends a pipeline from the pipeline pool based on the meta features from the database. When applying Meta-DPP, a user interface enables the data scientist, production expert, or IT expert to input their data set, learning task, ML algorithm and information about explainability. Given these four inputs, Meta-DPP provides a ranked recommendation of the DPP pipelines from the pool. Probabilities provided by the meta model further indicate how certain Meta-DPP is about the recommendation.

Verifying and validating revealed the correct development and implementation of Meta-DPP. The validation on 324 production use cases further prove that Meta-DPP outperform essential pipelines on average, whereby essential pipelines ensure the functioning of ML algorithms by performing minimum DPP. As a conclusion, the main research question was positively answered.

I Table of Contents

II Formula Symbols and Abbreviations

Formula symbol	Description
r^i	Target variable with i dimensions
w_i	Weight of node i
x^i	Input vector with i dimensions
x_i	Input of node i
$\hat{y}$	Predicted target
θ^*	Best parameter setting of ML algorithm
μ	Mean
$F1$	F1 score
FN	False negative
FP	False positive
MAE	Mean absolute error
MSE	Mean squared error
$RMSE$	Root mean squared error
TN	True negative
TP	True positive
X	Data set
b	Bias
d_{mr}	Distance between score of maximum and recommended pipeline
d_{re}	Distance between score of recommended and essential pipeline
d_{ue}	Distance between score of upper bound and essential pipeline
d_{ur}	Distance between score of upper bound and recommended pipeline
y	Actual target
θ	Parameter setting of ML algorithm

Formula symbol	Description
σ	Standard deviation
$\sigma(\cdot)$	Activation function

Abbreviation	Description
AdaB	Adaptive boosting (AdaBoost)
ADASYN	Adaptive synthetic sampling
AI	Artificial intelligence
ANN	Artificial neural network
APS	Air pressure system
AutoML	Automated machine learning
Bagging	Bootstrap aggregating
BFE	Backward feature elimination
Blisk	Blade integrated disk
BO	Bayesian optimization
BOHB	Bayesian optimization hyperband
Both	Data set with both missing values and categorical features
CAD	Computer-aided design
CAM	Computer-aided manufacturing
CART	Classification and regression tree
CASH	Combined algorithm selection and hyperparameter optimization
Cats	Data set with categorical features
CMM	Coordinate measuring machine
CNC	Computerized numerical control
CRISP-DM	Cross-industry standard process for data mining
DIN	Deutsche Industrienorm (”German industry norm“)
DPP	Data preprocessing
DPSO	Data pipeline and optimization
DSS	Decision support system
DT	Decision tree

Abbreviation	Description
E	Essential pipeline
ENN	Edited nearest neighbor
ET	Extra trees (extremely randomized trees)
FFS	Forward feature selection
GB	Gradient boosting
GIGO	Garbage in, garbage out
GNB	Gaussian Naïve Bayes
GP	Gaussian Process
HB	Hyperband
HCF	High correlation filter
HGB	Histogram gradient boosting
HPO	Hyperparameter optimization
ICA	Independent component analysis
ID	Identifier
IDA	Intelligent discovery assistants
IDSS	Intelligent decision support system
IoP	Internet of production
IQR	Interquartile range
ISO	International standard organization
ISOMAP	Isometric feature mapping
IT	Information technology
KDD	Knowledge discovery in databases
K-NN	K-nearest neighbor
LASSO	Least absolute shrinkage and selection operator
LB	Lower bound (worst performing pipeline of the pipeline pool)
LDA	Linear discriminant analysis
LGBM	Light gradient boosting machine

Abbreviation	Description
LLE	Locally linear embedding
LogReg	Logistic regression
MAR	Missing at random
MCAR	Missing completely at random
MDS	Multidimensional scaling
MICE	Multiple imputation by chained equations
ML	Machine learning
MLP	Multilayer perceptron
MNAR	Missing not at random
MV	Missing value
Mvs	Data set with missing values
NN	Neural network
Norm	Data set with no missing values and no categorical features
NRMSE	Normalized root mean squared error
OEE	Overall equipment effectiveness
OS	Operating system
PC	Principal component
PCA	Principal component analysis
PQ	Product quality prediction
Q	Quality
R	Recommended pipeline by the meta model
RAM	Random-access memory
RF	Random forest
RFE	Recursive feature elimination
RS	Random search
SEMMA	Sample, explore, modify, model, assess
SFE	Sequential feature elimination

Abbreviation	Description
SGD	Stochastic gradient descent
SICE	Single center imputation from multiple chained equations
SMAC	Sequential model-based algorithm configuration
SMBO	Sequential model-based optimization
SMOGN	Synthetic minority over-sampling technique for regression with gaussian noise
SMOTE	Synthetic minority oversampling
SMOTEENN	Synthetic minority oversampling edited nearest neighbor
SMOTENC	Synthetic minority oversampling nominal continuous
SMOTETomek	Synthetic minority oversampling tomek
SOM	Self-organizing map
SVC	Support vector classification
SVM	Support vector machine
SVR	Support vector regression
TPE	Tree parzen estimator
TPOT	Tree-based pipeline optimization tool
t-SNE	t-distributed stochastic neighbor embedding
UB	Upper bound (best performing pipeline of the pipeline pool)
XGBoost	Extreme gradient boosting

III List of Figures

IV List of Tables

1 Introduction

1.1 Initial Situation and Motivation

In the last decade, the production sector has been significantly shaped by developments towards Industry 4.0. Industry 4.0 aims at digitalizing the production by intelligent networking of machines, industrial processes, and people [BUND21]. Large amounts of data facilitate the targeted optimization of production by increasing quality, flexibility, as well as customer satisfaction. According to a German study from 2020, 59 % of surveyed companies already have specific applications in the area of Industry 4.0 in place [BITK20].

Nowadays, the modern era of production is accompanied by an ever-increasing availability and variety of data, which serves as a backbone for successful digitalization. For the operationalization of Industry 4.0, frameworks such as the *Internet of Production* (IoP) were developed, which attempts to enabling data-based decisions in production context-dependent and in high quality. The smart data layer represents one central element of the IoP, which analyzes previously aggregated data to facilitate automated decisions and context-specific decision support in order to eventually achieve big leaps in efficiency or quality. [SCHM20, p. 490]

Machine learning in production

To analyze aggregated data in production efficiently, machine learning (ML) became one of the most popular and common approaches. ML, as a subdomain and enabler for systems of artificial intelligence (AI), offers the potential to extract information from historical data by discovering complex patterns using more elaborate algorithms than common analytical approaches. [WEIC19, p. 1889] These properties allow a wide application of ML, such as for prediction, detection, or forecasting. [WUES16]

Numerous and versatile ML success stories can consequently be found in production. Prominent examples are the prediction of product quality, optimization of process designs through automated decisions in operating procedures, detection of anomalies within machining processes like milling or drilling, predictive maintenance to be capable of predicting tool wear during turning, or redesigning products based on customer feedbacks [KRAU19b]. Thereby, most use cases can be assigned to supervised learning and its corresponding tasks of either regression, i. e., predicting a continuous variable like tolerance deviation, or classification through, e. g., classifying whether a product is within or out of specification. The trend towards the use of ML is expected to continue. According to a research, a global growth can be expected by 14 % until 2030, while Germany's AI market will potentially grow by 430 billion € with manufacturing

among the biggest beneficiaries (52 billion €) [PWC17]. In addition, discussions about the increase of production sustainability are giving rise to further promising use cases for the application of ML through, for instance, sustainable designs of value chains [PWC20].

Although being widely applied, ML itself represents a highly interdisciplinary endeavor with very complex tasks and design decisions to be made during the creation of ML models. Figure 1-1 depicts the ML pipeline in production defined by FRYE ET AL. including its phases and subphases, while the phase data preprocessing is highlighted [FRYE21b]. Starting from the use case selection, data needs to be integrated and prepared for further modelling. Once a performant ML model is trained and tested, it can be deployed in the production environment and eventually certified. During the creation of performant ML models, different disciplines and competencies like production, IT and data science experts need to be involved, which are gaining mutual understanding of data and production process throughout the ML project.

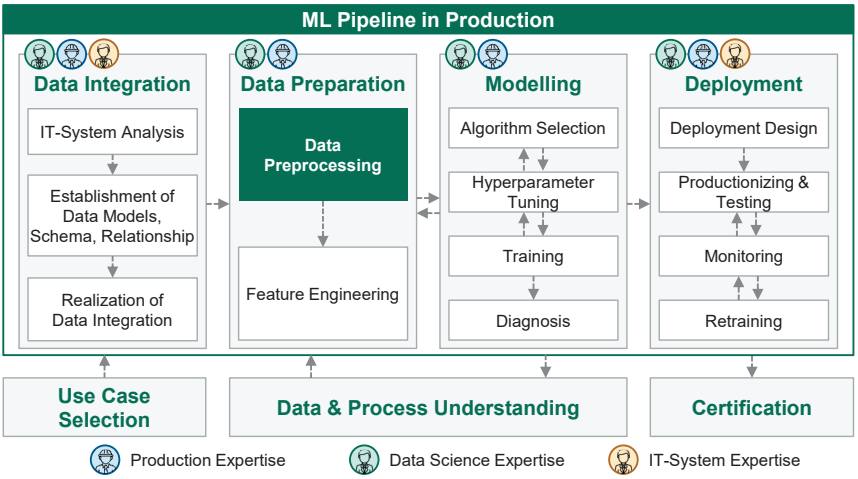


Figure 1-1: ML pipeline in production with focus on data preprocessing modified from [FRYE21b]

Data preprocessing

To achieve performant ML models in production, a high-quality data set is required, which is realized through data preprocessing (DPP). DPP represents the activity of transforming raw data of poor quality into a high-quality data set through the application of different DPP methods. Poor data quality negatively affects the quality of any analysis, which is denoted as “garbage in, garbage out”-principle. [GARC15, pp. 10-11] Consequently, messy data results in financial costs. According to Gartner’s data quality

market survey, the financial impact of poor data quality lies at around $ 15 million per year [GART18a]. Poor data quality has the potential to ruin the vision and whole idea of Industry 4.0, which is to support in data driven decision making. For ML, one of the main reasons why projects fail can be attributed to poor data quality. [GART18b] Based on these examples, it can be concluded that DPP is essential for company's success and reputation.

Raw tabular data generated in production exhibits very different kinds of quality issues, making DPP a complex and mainly manual undertaking, as DPP must be tailored to the quality issue at hand. For instance, in case of the presence of noise in form of missing data, outliers, or duplicates, data needs to be cleaned. Categorical features, skewed data, and data within different ranges require data transformation for the creation of performant ML models. Since data can be high-dimensional with correlated and unimportant features, data needs to be reduced. Data augmentation and balancing are applied in case of having too few instances or a class imbalance of the target variable in a classification task. [GARC15, p. 11] Ultimately, DPP can be divided into the steps of cleaning, transformation, reduction as well as augmentation and balancing [BURD20], [SLIJ20], [FRYE20]. In practice, multiple data quality issues need to be addressed by DPP resulting in DPP methods, which need to be interconnected appropriately into DPP pipelines. [FRYE20]

Selection of data preprocessing (DPP) pipelines in production

Despite the classification of DPP in respective steps, choosing DPP methods and configuring them into pipelines suitable for the specific use case is one of the fundamental challenges data scientists face in ML projects [PYLE99, p. 87]. The challenges can be classified by the *data set properties*, *ML algorithms*, high number of *available DPP methods*, and *conditionality* of DPP methods, as well as production characteristics.

Data set properties such as data set size influence the choice of DPP methods. For instance, missing values or categorical features highly influence the selection of DPP methods and its subsequent combination into a DPP pipeline. Here, choosing a DPP method to encode categorical columns has a large impact on the resulting data set and thus on ML performance. A tabular data set of a milling process serves as an example for showing the complexity of DPP, in which material to be cut can be either three different titanium, or two different nickel alloys. Applying a *OneHotEncoder* results in a data set with an increase of four additional, sparse columns, while a *LabelEncoder* retains the number of columns by replacing categories through ordered numbering. While the increase of features may result in a too sparse data set and poor

ML model performance, encoding nominal categories in an ordered manner yields to misinterpretations by the ML algorithm. [POTD17, p. 7]

Besides data set characteristics, *ML algorithms* place requirements on DPP. For instance, *decision tree* (*DT*) algorithms do not perform well in the presence of missing values (MVs), making it important to handle MVs prior to ML model training [KHOS20]. *Artificial neural networks* (*ANNs*) require only numerical data and cannot be executed in case of categorical columns [GARC15, p. 4]. In practice, there are *hundreds of DPP methods* from which the most suitable ones need to be implemented. Different DPP methods intended to aim for the same purpose may lead to different ML performance as shown in the milling example. The complexity and selection possibilities increase further when combining methods to DPP pipelines. [GÉRO18, p. 36] At the same time, *conditionality*, and mutual influence of different DPP methods need to be considered during the configuration of DPP pipelines. For instance, when selecting a *principal component analysis* (*PCA*) for reducing data set's dimensionality, data needs to be scaled beforehand [PULA20]. Lastly, as for ML algorithms, DPP methods are dependent on their hyperparameters, which needs to be set prior to the execution of the pipeline.

The production domain place further requirements on DPP. One crucial requirement is the necessity of overall performant DPP pipelines to be executed for production use cases. Computing times should further be within a manageable amount of time and DPP methods being applied may be explainable. In addition, the computing platform being used in terms of operating system, number of cores and random-access memory (RAM) available pose requirements, which should be considered when selecting DPP pipelines for a given production use case. [GARC15, pp. 3-5]

Due to manifold challenges, DPP is nowadays performed manually via trial and error by data scientists, yielding to the fact that DPP accounts for approximately 80 % of the time needed to conduct ML projects. [GABE17] Identifying DPP pipelines is additionally challenging for production experts, as they do not possess knowledge and expertise about determining suitable DPP pipelines. For these reasons, a decision support system is required, which recommends most suitable DPP pipelines for given production use cases.

As of now, many approaches have been introduced that aims at supporting the selection of DPP methods or even pipelines. Approaches developed by Bilalli et al. or Berti-Equille support the identification of DPP methods and pipelines [BILA18], [BERT19]. However, the assessed DPP methods are often limited to individual DPP steps, and do not consider production-specific requirements. Therefore, a decision

support system is needed, which proposes most suitable DPP pipelines for production use cases.

1.2 Problem Statement, Thesis Goal, and Research Questions

DPP fundamentally influences the quality of the data set as well as ML model performance, while being a tedious and non-reproducible task. The process of identifying suitable DPP methods and combining them into DPP pipelines poses a major challenge. Thus, the practical problem can be stated as the unavailability of a system that supports data scientists and production experts in selecting suitable DPP pipelines depending on production use case requirements. From the practical problem, the theoretical and scientific need can be derived, which comprises a decision support system that recommends suitable DPP pipelines given the characteristics of production use cases.

Therefore, the goal of this thesis is the development of a decision support system for recommending DPP pipelines for ML applications in production. Consequently, the main research question (MRQ) can be formulated as follows:

MRQ Can a decision support system be developed that supports in recommending DPP pipelines for ML applications in production?

To answer the main research question, the following sub research questions (SRQ) are derived and need to be answered. Respective chapters of this thesis are provided in brackets:

SRQ 1 Which requirements need to be met by the decision support system for recommending DPP pipelines? (Chapter 3.1)

SRQ 2 Which approaches and technologies can serve as a basis for the development of a decision support system? (Chapter 3.2)

SRQ 3 Which core components can form a decision support system for recommending DPP pipelines in production (Chapter 4.1)?

SRQ 4 Which DPP methods are most suitable for preprocessing production data and how can DPP methods suitably be configured into DPP pipelines (Chapter 4.2)?

SRQ 5 How can the components of the decision support system be set up? (Chapter 4.2, Chapter 4.2.4, Chapter 4.4)

1.3 Research Approach

The research approach in this thesis can be classified in the scientific landscape provided by ULRICH [ULRI76, p. 305] Referring to Figure 1-2, ULRICH distinguishes between formal and empirical science. Formal sciences focus on the construction of sign systems, to which philosophy, logic, or mathematics contribute to. Empirical sciences can either follow a theoretical or practical goal. Theoretical procedures can be assigned to fundamental sciences, in which empirical details of reality are explained.

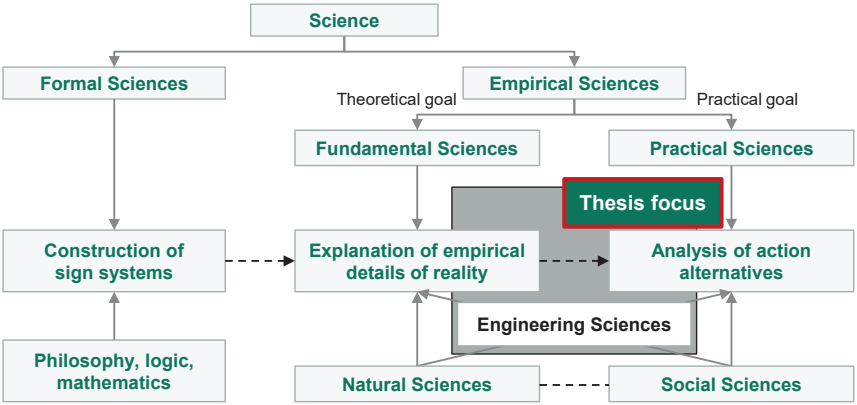


Figure 1-2: Classification of sciences according to ULRICH [ULRI76, p. 305]

When following a practical goal, practical or applied sciences analyze human action alternatives. Since fundamental and practical sciences cannot be clearly separated in practice, engineering sciences can be assigned between the two research fields as engineering sciences contribute to both basic and application-oriented research, involving both natural and social sciences. [ULRI76, p. 305] This thesis can be assigned to engineering science, while focusing on applied science, which is built on technically oriented basis [CZIC08, p. 13]. Since the goal of this work is to develop a decision support system for recommending DPP pipelines, the research stems from a practical context, as it is obligatory for practical sciences [ULRI81, p. 10]. In addition, the thesis aims at solving decision-making problems through knowledge extraction, which represents the most important objective in applied sciences according to KUBICEK [KUBI77, p. 5].

The thesis structure refers to the seven steps of ULRICH'S research approach, which is shown in Figure 1-3 [ULRI81, p. 20]. As a first step, the practical problem is determined, which is covered in the introduction and detailed in the fundamentals of this thesis Chapter 1 and Chapter 2. From there, the main research question is derived and confirmed. Afterwards, according to ULRICH, problem-related theories are interpreted as

well as hypotheses from both empirical fundamental sciences identified, which serve as basis for classifying and specifying the practical problem. These activities are covered in Chapter 2, in which fundamentals as well as requirements in the recommendation of DPP pipelines for ML applications in production are discussed. Additionally, problem-related approaches to solve the practical problem are specified and distinguished from the present thesis in Chapter 3, based on which the further actions are deduced.

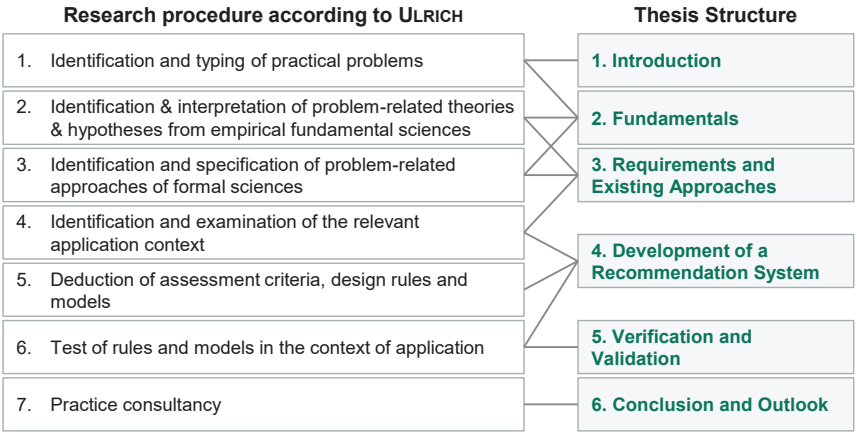


Figure 1-3: Thesis structure according to the research approach by Ulrich

Based on the resulting need for research (Chapter 3.2) and the requirements (Chapter 3.1), the decision support system for proposing suitable DPP pipelines in production is developed in Chapter 4 that is capable of being implemented in real applications in production. In particular, the main chapter proceeds according to the method of action research introduced by LEWIN, which states that results obtained are already considered during development and put into practice [LEWI46]. In this regard, the development of the decision support system is characterized by the continuous acquisition of information, constant scrutinizing, and practical actions. Through the continuous creation of data within the development, knowledge is gained, which serves the development of the decision support system and thus generates implementable solutions for the practical problem (Chapter 4). [MAYR02, pp. 51-54] After the development, the decision support system is verified and validated in the context of the application and optimized based on the findings (Chapter 5). Then, the results of the decision support system are critically discussed in Chapter 6. The thesis ends with a summary and an outlook providing practical consultancy to the science community (Chapter 7).

2 Fundamentals in the Recommendation of DPP Pipelines for ML Applications

The fundamentals in the recommendation of DPP pipelines for ML applications in production first classifies different applications and use cases of ML in production (Chapter 2.1). Subsequently, existing process models for conducting ML projects are discussed, in which DPP is classified (Chapter 2.2). ML is further introduced, in which the robustness of ML algorithms against data set properties is presented (Chapter 2.3). A deep dive is made into DPP, underlining the importance of DPP, introducing production data characteristics and its requirements on DPP. In addition, DPP methods are detailed and the creation of DPP pipelines discussed (Chapter 2.4). Chapter 2.5 deals with the fundamentals of decision support systems to be used or adopted for the recommendation of DPP pipelines. The chapter ends with an interim conclusion (Chapter 2.6).

2.1 Machine Learning Applications in Production

In general, many versatile ML applications and use cases exist in production. The different use cases can be classified in application areas and corresponding applications according to Figure 2-1, which is modified and detailed from KRAUß ET AL. and based on several surveys [GEIS17], [TATA14], [GURS16], [HARD06], [MCKI17], [MCKI16], [TATA13], [WANG18], [KRAU19b], [WUES16]. The application areas can be divided into machines and assets, product, and process. In case of the application area *process*, applications of process design, process management, routing & scheduling, as well as predictive process control can be assigned. Exemplary use cases are provided for every application. For instance, routing and scheduling can be optimized through the ML based prediction of lead times, while the decision about repairing or buying new spare parts can be automated using ML [LANG18], [GYUL18]. In case of the application area *machines and assets*, system and machine anomalies, or machine faults can be classified, and tool wear predicted [LIND19]. *Products* can be further developed based on customer feedback [ZHAN21].

In practice, the overview shown is subject to constant expansion due to the broad and increasing use of ML [PWC20]. Political and social developments substantially influence the production sector. The urge towards more sustainability forces the creation of new products. Wind turbines, photovoltaic systems, electric or hydrogen-based mobility can only be realized in case of highly scaled production. ML is a key enabler for optimizing and scaling up whole productions and is nowadays already considered in the development phase of new factories. [MCKI21], [HARV21] Consequently, ML use

cases are about to increase in upcoming years and Figure 2-1 only provides an excerpt of real-world applications in production. The ML use cases can thereby be based on different data types, which can be, for instance, tabular, image, or text data. Due to the high amount of use cases in production with tabular data, the focus lies on tabular data in the remainder of this thesis.

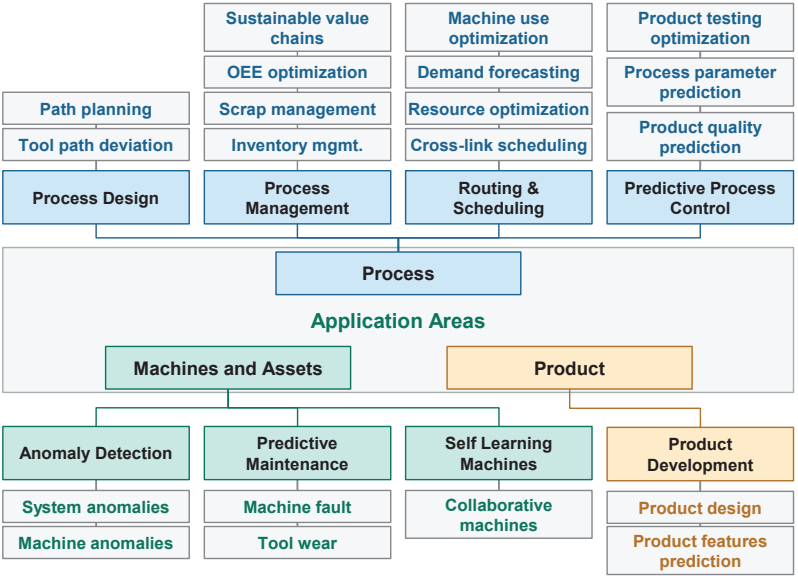


Figure 2-1: Application areas, applications, and exemplary use cases of ML in production according to [KRAU19b]

Different use cases, which can be seen in Figure 2-1, can also be combined. Product quality prediction can be used for an optimization of routing and scheduling. The output of the ML model to forecast the expected quality can trigger a rescheduling, based on which machine's overall equipment effectiveness (OEE) can be predicted for future products. [FRYE21c]

Many different ML use cases can also be realized within only one production use case. Therefore, a production use case is presented in the following, which illustrates the diverse applicability of ML but also focuses on four representative and very common ML use cases in production. In addition, the importance of DPP is already outlined. The milling process of blade integrated disks ("Blisk") is used, in which ML use cases are shown that can be also transferred to various production use cases. Blisks are used in high- and low-pressure compressors of aircraft engines. Conventionally, Blisks are produced using a 5-axis CNC milling machine. As a first step, the machine program

is created and optimized through planning using computer aided manufacturing (CAM). Secondly, the work piece, which has previously been turned to size, is machined in a milling process, and thirdly inspected on the coordinate measuring machine (CMM) to determine the profile deviations of the Blisk. Figure 2-2 summarizes the process schematically. Four use cases can be derived underlining the potentials of ML in production: process design through the prediction of tool path deviation (1), predictive maintenance through the prediction of the remaining useful lifetime of the tool (2), prediction of process parameters by forecasting the sound pressure level during cutting (3), and product quality prediction by predicting the profile deviation using process data (4).

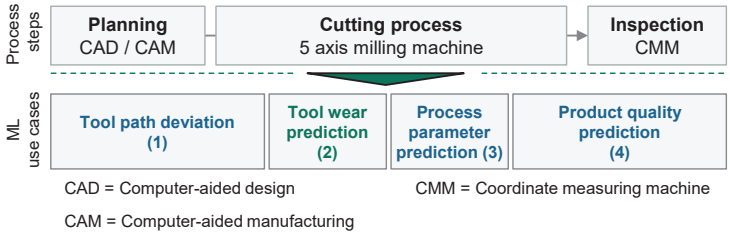


Figure 2-2: Representative ML use cases assigned to process steps of Blisk milling

(1) Tool path deviation (process design – process)

The adherence to the tool path during milling determines tolerance deviations as well as process stability. During CAM planning, the tool path is defined so that maximum quality can be ensured during machining. However, inaccuracies occur during machining due to tool deflection, workpiece deflection, and vibration. Therefore, the goal of the ML application is to predict the deviation of the tool path based on planning data to adjust the output parameters already in the planning phase in such a way to ensure high quality. [ICTM21] Tool path compensation strategies can thereby be found in literature [AKRI19].

Independent variables consist of the axis data and milling setting parameters, which are in the work piece coordinate system. The target variable is the toolpath deviation, which is calculated from the axis data in the machine coordinate system. In addition, machine parameters are sampled at a different frequency than is available through CAM planning. In the prediction of tool path deviation, the features of CAM planning are in different scales, while high number of features need to be selected.

(2) Tool wear prediction (predictive maintenance – machines and assets)

During milling, the tool is subject to continuous wear. Experiments have shown that tool wear has a significant influence on the final quality of the product. Tool wear increases degressively at the beginning, then linearly and progressively at the end of the lifetime as illustrated in Figure 2-3 [KLOC08, p. 263]. At the end, tool breakage occurs. While linear tool wear is natural and enables a stable process, severe tool wear at the end of the service life has a poor influence on the final quality. Therefore, it is desirable to be able to determine when the tool needs to be replaced in advance to ensure quality and increase efficiency. [HOLS21], [VENK14]

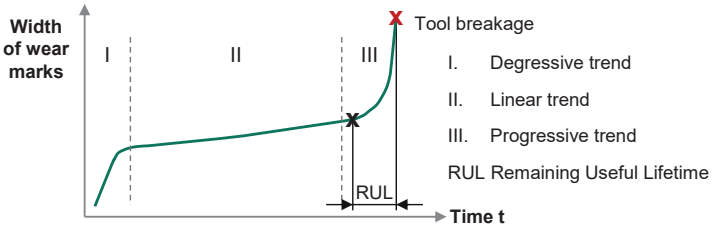


Figure 2-3: Schematic representation of the tool life according to [KLOC08, p. 263] and remaining useful lifetime (RUL)

Based on the process data of the milling process, the tool wear can be concluded. In practice, the wear mark width is measured to determine the tool wear. The challenge with this application is that only very few events of high tool wear are available, which are to be matched with a high amount of process data. This imbalance of data is a common problem with production data. [FRYE20]

(3) Process parameter prediction (predictive process control – process)

During milling, if a multiple of the natural frequency of the component is met, vibrations can lead to large profile deviations and poor surface finishes. Figure 2-4 illustrates the prediction of the sound pressure level to capture vibrations.

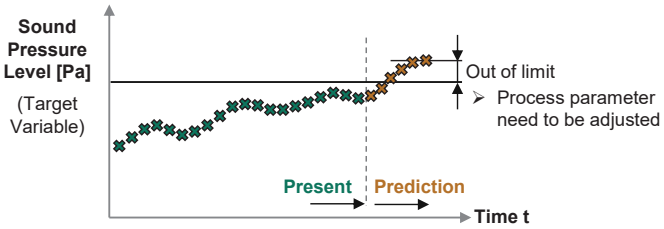


Figure 2-4: Illustration of prediction of process parameters, here sound pressure level

Vibrations can be detected via microphones or acceleration sensors and prevented by adjusting machine parameters, such as spindle speed or feed rate. The aim is therefore to predict vibrations during the process, based on which machine parameters can be adjusted. The target variable is then the sound pressure level, which is recorded over time. [SAAD18], [CHEN17], [VENK14]

To capture the vibrations, a frequency needs to be acquired that is significantly higher than the frequency of the machine sampling rate. This results in the machine sampling at a frequency of 100 Hz, while the microphone samples at a frequency of 25 kHz. This means that the machine data must be augmented to the frequency of the microphone or the frequency decreased to the machine data [FRYE19].

(4) Product quality prediction (predictive process control – process)

In practice, measuring the profile deviations of milled blisks takes several hours. Acquired process data during milling, such as accelerations or axis data, can provide information about profile deviations. Therefore, the goal is to predict profile deviations based on process data and thus to inspect only those areas that show anomalies in the process data. [CHIU17], [PFIR19], [CHEN17]

In this case, input features are process variables such as accelerations, which are available in machine coordinate system. On the contrary, the dependent variable represents the profile deviation in the CMM in a different coordinate system. Therefore, data needs to be transformed in a unique coordinate system to predict the product quality.

Interim conclusion

The presented ML use cases for Blisk milling can be transferred to further processes such as milling, turning, drilling, or electrical discharge machining. For these use cases, several data quality issues can be found that need to be addressed during DPP. [WUES16], [KÖKS11], [WEIC19], [KIM18] The presented use cases consider tabular data sets, while image or text data can also be found in production. However, in this work, the focus is put on tabular data sets due to the high number of ML use cases in production based on tabular production data from processes, machines, or quality inspections. To implement ML use cases successfully in production, process models have been developed, which are presented in Chapter 2.2.

2.2 Process Models for Conducting Machine Learning Projects

To successfully conduct data-driven projects in general, generic guidelines have been developed in the past that show the process of extracting knowledge from data in a

stepwise approach. Most known frameworks range from *knowledge discovery in databases* (KDD), over *sample, explore, modify, model, and assess* (SEMMA) to *cross-industry standard process for data mining* (CRISP-DM). [FAYY96], [CHAP00], [SAS22] In total, all approaches exhibit generic approaches, which are less tailored to production characteristics.

The key role in the implementation of ML projects is assigned to data scientists. While data scientists interpret data by using statistical methods to assist within a business and its decision making process, further expertise is demanded particularly in production [DHAR13]. Domain experts support the data scientist during the creation of ML models and especially DPP. Here, the support of process experts is manifold:

- Interpretation of process, product, and production data
- Identification of most suitable features based on their experience
- Support in the conjunction of different data sources
- Validation and interpretation of data analysis results
- Implicit guidance of ML process due to the requirement of using transparent DPP methods and ML algorithms
- User-centric view, i. e., providing a direction on how data and results should be stored and saved for an efficient use on the long term

To consider the production related requirements and different expertise, the ML pipeline referring to Figure 1-1 has been developed. The ML pipeline for production consists of use case selection, integration and preparation of data, modelling, deployment, and subsequent certification. In the following, all steps are briefly introduced highlighting different expertise that are required according to FRYE ET AL. [FRYE21B].

ML use case selection

The first step of a ML project consists of the selection of suitable use cases. By considering profitability, strategic relevance, stakeholder commitment of the use case, existing infrastructure at hand, and data set, suitable ML use cases can be prioritized. Different stakeholders are already involved within the use case selection to determine the feasibility of corresponding use cases. The output are one or more selected ML use cases.

Data integration

Starting from prioritized ML use cases, data in general needs to be integrated from different sources. Combining data from different sources requires experts from IT, production and data science. IT experts possess the knowledge about general IT architecture, a production expert knows which data needs to be used, and the data scientist knows how the data should be used. Therefore, the first step of integrating data is the

analysis of IT systems at hand, based on which data models, schema and relationships are established. In case the sources contain different frequencies and time stamps, data also needs to be synchronized prior to data preparation [FRYE20]. Ultimately, data integration is realized.

Data preparation

Before modelling, data needs to be prepared to create a high-quality data set. Integrated data usually still consist of versatile data quality issues such as missing values or duplicated rows that needs to be addressed during data preparation. Preparing data can be subdivided into data preprocessing (DPP) and feature engineering. While preprocessing data consists of the manipulation of data using DPP methods to achieve a high-quality data set, feature engineering focuses on feature creation through production expertise. As mentioned in Chapter 1.1, the focus in this thesis lies on supporting different disciplines, i. e., production experts as well as data scientists, in the selection of DPP methods. The output of data preparation is a high-quality data set ready for modelling.

Modelling

Based on a high-quality data set, suitable ML algorithms are to be selected. Production experts influence the choice of algorithms in terms of transparency, complexity, and computing times. Further, the selection of ML algorithms is mainly driven by the learning approach and learning task at hand, requirement of performance, speed and explainability [KRAU19a]. During training, ML algorithm's hyperparameters are tuned to fit the model to the data. The performance of the ML model is eventually assessed on test data by corresponding metrics. In practice, the steps of data preparation and modelling are highly iterative. After a basic data preparation, baseline ML algorithms are trained to identify the complexity of the underlying problem. Afterwards, DPP is extended, and more sophisticated ML algorithms are implemented if required. Within data preparation and modelling, mutual information exchange is fostered and necessary between the different experts to achieve best performing results, which is represented by data and process understanding in Figure 1-1. The output of the modelling phase represents a performant ML model.

Deployment

Performant ML models only add value to manufacturing companies if they are deployed and operated in production. In a first subphase, the deployment needs to be designed since ML models can be used for automated decision-making, but also for supporting decisions on the shop floor. Several ML models can be deployed dependent on the use case, in a shadow, canary release, online, or competing design [SAMI19b].

Subsequently, the software system including ML models can be productionized and tested following the methodology provided by DevOps [KIM16]. After deployment and during operation, the ML model is prone to degradation due to data drift. Therefore, monitoring the software system and its embedded ML models need to be ensured. Based on retraining mechanism, for instance, scheduled or based on a degradation threshold, the performance of ML models can further be optimized or retained. [SAMI19a] Output of this phase is the deployed software system with embedded ML models in operation.

Certification

Lastly, the software system including ML models can be certified. Certification comprises the determination of robustness in case of new input data within same distributions and corresponding similar output, explainability of ML model decisions, robustness if new input data exhibits different data distribution [MOLN22]. Certifying ML systems also require the contact of relevant certification organizations that accompany in the development of a certification strategy and conducts pre and final audits for certifying ML systems eventually. [BRAN19]

Consequently, the implementation of ML in production follows a standard procedure on a high level with complex tasks, iterative steps, and many disciplines. Since the focus of this thesis lies on preprocessing tabular data for ML applications, ML, its algorithms, and the robustness of ML algorithms against data set properties are discussed in detail in Chapter 2.3.

2.3 Machine Learning

The principle of ML, “[…] which gives computers the ability to learn without being explicitly programmed” was first introduced and spread by Arthur Lee Samuel in 1959 [SAMU59, pp. 210-229]. While an algorithm executes deterministic instructions based on input data to generate an output, ML algorithms create mathematical models for generalizing data patterns, or extract knowledge by automatically improving performance through experience [MITC97], [SUTH16, p. 123].

2.3.1 Machine Learning Approaches

In general, ML can be subdivided in three different types defined by the learning approach: supervised, unsupervised, and reinforcement learning [WUES16, p. 10]. The three categories are described in more detail in the following.

Supervised learning

In supervised learning, input-output pairs are available, in which ML algorithms predict labels of the data set. A set of independent variables (x), called features, attributes, or dimensions, represents the input, while the output is the dependent variable (y), which is also called label or target variable. Supervised learning is characterized through the knowledge about the label. [GÉRO18, p. 38] ML algorithms in supervised learning then aims at determining the best possible approximation of the function $f(x) = y$ [RUSS10, p. 695]. Ultimately, the performance of trained ML models is assessed using new unseen test data. In production, supervised learning algorithms are most often applied. [WUES16]

Unsupervised learning

If no target variable is given when training ML algorithms, the approach can be assigned to unsupervised learning. Unsupervised learning aims at grouping existing data to extract information and patterns. [HAN12, p. 25] To group data successfully, clustering algorithms are commonly used, which are especially suitable for visualization purposes [BISH06, p. 3], [AWAD15, p. 4]. According to GE ET AL., supervised and unsupervised learning are most applications in the industrial sector covering 80 – 90 % of all applications [GE17, p. 20594].

Reinforcement learning

In reinforcement learning, an agent attempts to successfully proceed in an environment by a set of actions. Depending on the success, the agent receives positive answers, denoted as reward, or a punishment in case of negative feedback. Through learning a strategy to optimize his set of actions within the environment, the agent aims at maximizing the reward. [MITC97], [FLAC12, p. 360] For production, reinforcement learning is mainly applied to the field of robotics [GE17]. However, new applications can be found in the field of routing and scheduling [WASC18].

Since supervised learning is commonly applied, the thesis focuses on supervised learning algorithms for the ML applications in production, which will be detailed in the following subchapter. However, DPP methods can also comprise unsupervised learning approaches, which will be detailed in Chapter 2.4.3.

2.3.2 Supervised Learning

In the following, supervised learning is classified. Secondly, the training process and cost functions dependent on corresponding learning tasks are described before introducing the training algorithm for supervised learning.

Learning tasks in supervised learning

In supervised learning, ML algorithms are trained to generalize on previously unseen data by discovering patterns between input-output pairs [BISH06]. The tasks within supervised learning can be distinguished into either classification or regression, which are represented in Figure 2-5 and described in detail below [SMOL08, pp. 9-11].

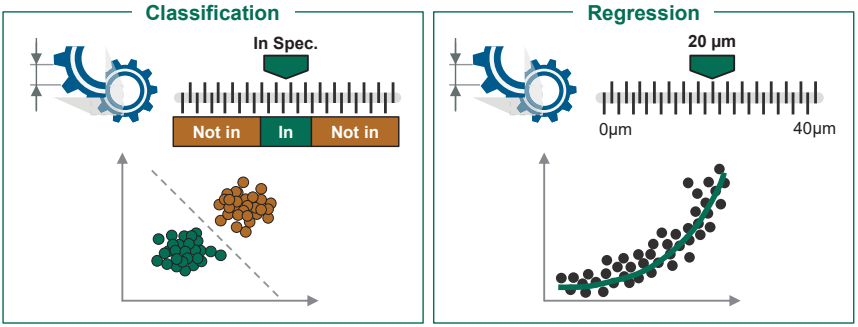


Figure 2-5: Supervised learning tasks: classification (left), regression (right)

- In *classification*, labels are part of a finite number of discrete categories. The challenge lies in the identification to which category a given observation of dependent variables belongs. If the target variable contains two categories, a binary classification is present. A multi class classification needs to be performed, if more than two categories are available in the target feature. [GÉRO18] A typical classification use case can be found in the product quality prediction, as detailed for the Blisk use case in Chapter 2.1. Here, the product can either be within or out of specification, representing a two-class classification. In practice, since profile deviation can be classified in more categories than good or bad, a multi class classification is followed.

- *Regression* comprises the task of estimating a continuous, real-valued variable based on input features. [SMOL08, pp. 9-11] According to the Blisk example in Chapter 2.1, the prediction of the remaining useful lifetime of the cutting tool represents a regression task, since the output, e. g., 7 cycles or 10.52 min, exhibits continuous data. In addition, regression tasks can also be transferred into a classification task by classifying the output using pre-defined threshold values.

Introduction of training, validation and testing

Prior to training, the entire data set needs to be split into a training and a test data set. Typical splits are 70 % until 80 % of training data, and 30 % or 20 % of test data, respectively. [MURP12, p. 23] Thereby, the test data is unseen during training to identify ML model performance in real applications. The entire process is depicted in Figure 2-6.

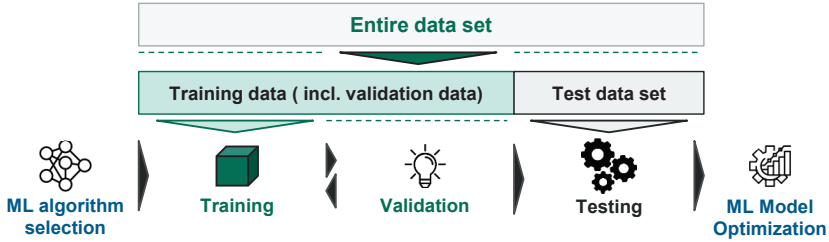


Figure 2-6: Principle of training, validation, and testing in supervised learning

For training, a suitable ML algorithm needs to be selected under consideration of speed, explainability, and performance [KRAU19a, pp. 52-54]. Within training, intermediate assessments of the ML model can be conducted through validation. Common approaches are *k-fold cross validation* or *leave one out cross validation*. [WITT05, p. 150] The resulting performance of the trained ML model is tested. Based on the ML model performance, the model can further be optimized. [FLAC12, p. 349].

Bias variance tradeoff

When selecting ML algorithms and given the quality of the data set at hand, the bias variance tradeoff needs to be considered. Figure 2-7 illustrates the bias variance tradeoff based on different complexities of trained ML models.

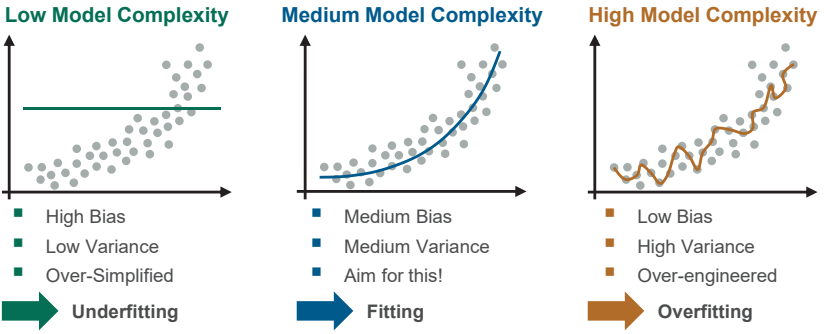


Figure 2-7: Visualization of the bias variance tradeoff

In case ML algorithms are too complex, the bias would be low, and variance of the model would be high. This scenario is called overfitting. In the situation, where variance can be low, and bias high, underfitting appears. The goal is to find ML models that generalize well on the data. [KOHA96] Besides the complexity of ML models, data quality plays a crucial role to prevent over- and underfitting during modelling. Noise in the data influences how a model generalizes on the data. In addition, the bias variance tradeoff shows the dependencies between data quality, DPP, and ML algorithm selection. This impact is discussed in detail in Chapter 2.4.3 and Chapter 2.4.4.

Loss functions and training ML algorithms

Training supervised learning algorithms represent an optimization problem that can be defined mathematically. Based on the input features and the target variable, a model is to be found, which minimizes an approximation error. If X is the data set with $X = \{x^i, r^i\}_{i=1}^N$, then $x^i \in \mathbb{R}^M$ represents the input vector of M dimensions and r^i is the target value, while i represents the instance from N instances in the data set. Given x^i, ML algorithms train a model aiming at identifying patterns to depict the relationship between x^i and r^i by generating the prediction $\hat{r}^i = f(x^i|\theta)$. The parameter θ represents the parameters of the trained ML model. To define the performance of an ML model, a loss function $Loss()$ needs to be defined, which uses the actual output and prediction of the ML model. When applying the loss function, the difference between the actual output and prediction, i. e., the $Error()$, can be determined. Ultimately, the loss can be calculated given the model and input data set as stated in Equation (2-1). [DEIS21, p. 260], [ALPA14, pp. 21-27]

$$Error(\theta|X) = \sum_i Loss(r^i, f(x^i|\theta)) \tag{2-1}$$

Due to the assumption of statistically independent input parameters x^i, the loss is averaged over N training examples. For classification, the most common loss function is the cross-entropy loss, which increases if the predicted probability diverges from the actual label. Thereby, cross-entropy, log loss, and negative log-likelihood are used interchangeably in classification tasks. [MURP12, p. 246] The lower the discrepancy, the higher is the learning progress of the model [SMOL08, p. 91]. In case of regression, mean squared error, quadratic loss or L2 loss are most often applied as loss function. Mean squared error is calculated through averaging squared deviations of actual outputs with predictions, while covering error magnitude, but ignoring its directions. Outliers are disproportionally high penalized due to squaring deviations. [BISH06, pp. 46-48] Ultimately, by minimizing the corresponding loss function of either classification or

regression, the optimal parameters $\theta^*$of the ML model are found in an iterative training process according to Equation (2-2).

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, Error(\theta|X) \tag{2-2}$$

To minimize the loss function and, thus, to find optimal model parameters, different training strategies are followed that are given by the nature of the underlying ML algorithm. The most common training algorithm is the gradient descent, which aims at optimizing the parameters during training by iteratively minimizing the error of the ML model using the loss function. Thus, partial derivates are calculated to move in the direction of most negative slope, i. e., to the minimum of the loss. [GOOD16, pp. 82-86] Besides mini-batch or batch gradient descent, which involves the sub sets or the whole data set, *stochastic gradient descent* (*SGD*) only selects one instance per iteration randomly. *SGD* exhibits less computing time for larger data sets. [GÉRO18, p. 119] In addition to the gradient decent, ML algorithms such as *decision tree* apply specific metrics like information gain or Gini index to split the input features to reduce the loss. Lastly, ML algorithms do not require training in case of the application of deterministic computations as it applies for *k-nearest neighbor* [WITT05, pp. 235-236]. Every training has in common that the exploitation can be stuck in local optima. To be able to also explore the search space in better optima, hyperparameter optimization can be applied [FEUR19, p. 9].

Hyperparameter optimization

Within the training phase of ML algorithms, hyperparameters can be optimized. Every ML algorithm contains hyperparameters. For instance, the number of hidden layers or number of neurons per layer represent two hyperparameters of a *multilayer perceptron*. According to the *no free lunch theorem* by WOLPERT & MACREADY, no ML algorithm performs universally best to achieve the lowest error [WOLP97]. This also means that there is not one hyperparameter setting per ML algorithm that performs universally best. By applying cross validation, different hyperparameter settings can be tried within training to increase the performance of the ML model. Many techniques have been introduced for hyperparameter optimization (HPO) that can be distinguished in four groups (see Figure 2-8). [FEUR19, pp. 3-18]

HPO can be performed through a model-free approach that carries out a full evaluation over a previously defined grid. Basic methods are grid and random search, but also biologically inspired algorithms such as ant colony optimization [BERG12]. When searching over the whole configuration space is too costly, approximation based algorithms such as successive halving or hyperband can be followed, which uses a portion of the whole budget, e. g., computation time, to identify best models and increases the

budget only on good performing ones. Besides model free approaches, *Bayesian optimization* can be applied, in which a surrogate model is used to propose the most suitable configuration that achieves the highest information gain on the configuration space. Examples for surrogates are *Gaussian process*, *random forest*, or *tree parzen estimator*. [FEUR19, pp. 9-13] Implementation such as *sequential model-based optimization* (*SMBO*) and *sequential model-based algorithm configuration* (*SMAC*) will be discussed in Chapter 3.2.3. When model-based and approximation methods are combined, a surrogate model recommends a set of configuration, in which the set is trained on different budgets. [FEUR19, pp. 14-18]

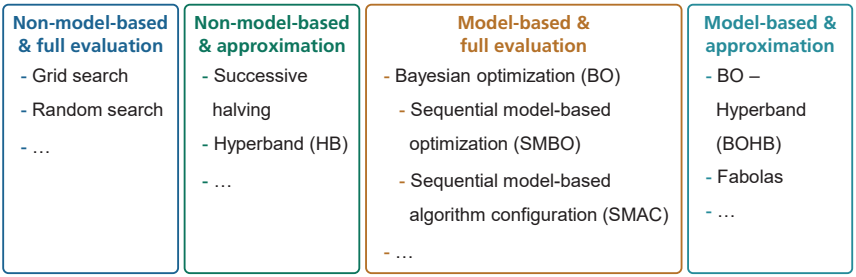


Figure 2-8: Classification of hyperparameter optimization techniques (excerpt taken from [KRAU22, p. 40])

Performance metrics

To assess the performance of ML algorithms, different metrics are used for classification and regression. The basis for the assessment builds the confusion matrix, which refers the predictions of the model to the actual label. [SOKO09, pp. 428-430] The main components of the confusion matrix are briefly described and explained based on the production use case of predicting the product quality in Blisk milling (see Chapter 2.1):

- *True Positives* (TP): Actually positive cases are also predicted as positive cases by the ML model. The product quality is predicted as being within specification, which is also the case for the actual component.
- *True Negative* (TN): Actually negative cases are also predicted as negative cases by the ML model. The ML model predicts that the product is out of specification, which is true.
- *False Positives* (FP): Actually negative cases are wrongly assigned as positive. Although the product is out of specification, the model concludes that the component is still within tolerances (also denoted as type 1 error)

- *False Negatives* (FN): Actually positive cases are wrongly classified as negative. The product quality is within specification, however, the ML model predicts the opposite (also known as type 2 error)

Based on the confusion matrix, different metrics can be derived, which are particularly suitable dependent on the production use case at hand. As the most basic metric and following Equation (2-3), accuracy calculates the share of correctly assigned classes respective to the entire data set. [BEKK13, pp. 27-29] Since $accuracy$ is prone to provide misleading results especially in case of imbalanced target variables, further metrics are used to assess the performance of ML models.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2-3}$$

$Precision$ represents the accuracy of positive cases, while $recall$ describes the true positive rate, also known as sensitivity. Besides general F scores, the $F1\ score$ combines both metrics through simultaneously determining the precision of the classification and the sensitivity of the minority class, as stated in Equation (2-4). [GÉRO18, pp. 88-90]

$$F1\ score = 2 \cdot \frac{Precision\ \cdot Recall}{Precision + Recall}, where$$
$$Precision = \frac{TP}{TP + FP}\ and\ Recall = \frac{TP}{TP + FN} \tag{2-4}$$

Since continuous values are predicted in regression, the assessment of the performance of a model is conducted by measuring distances between the actual target data and predicted values. Therefore, residues are calculated that subtracts the predicted value $\hat{y}$ from the actual value y. Different metrics can be derived from the subtraction. Commonly used are the *mean absolute error* (MAE), *mean squared error* (MSE) and *root mean squared error* $(RMSE)$. MAE calculates the absolute of the subtraction of $\hat{y} - y$. MSE and $RMSE$ consider larger errors in case of outliers and are therefore suitable for production use cases to prevent overfitting and penalize outliers. Equation (2-5) shows the mathematical formulation of $RMSE$. [WITT05, pp. 177-179], [KELL15] In practice, regression metrics are normalized based on different distribution measures of the continuous target variable for better comparability. Examples for normalizing the RMSE are the mean, range, or standard deviation. For identifying the performance of ML algorithms, a standard deviation reveals, whether a model is capable of making more precise prediction than forecasting within the variation of target values.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}} \quad (2\text{-}5)$$

Depending on the learning task and use case at hand, different ML algorithms can be selected. Due to the *no free lunch theorem* as stated above, numerous ML algorithms exist, which will be introduced in the following sub chapter. Since the focus in this thesis lies on recommending DPP pipelines for ML applications, a focus is put on the robustness of ML algorithms against data set properties.

2.3.3 Robustness of ML Algorithms Against Data Set Properties

As mentioned in Chapter 2.2, DPP and modelling are highly iterative, whereas ML pipelines mainly contain DPP, ML algorithm selection, hyperparameter tuning and ML model validation [TRUO19, p. 1472], [OLSO16a, pp. 1-3]. DPP is essential, since the performance of trained ML models highly depend on data set properties [GARC15, pp. 3-10]. The robustness of ML algorithms represents the degree on how ML algorithms perform in the presence of certain data set characteristics and data quality issues. [YURC21], [FERN05, pp. 333-335] In addition, high dimensional data, imbalanced classes in the label, input data in different ranges, and skewed input features affect the ML model performance. In case of either categorical features or missing values, many ML algorithms do not work at all. [GARC15, pp. 46-54]

In conclusion, due to different levels of robustness against data quality issues, ML algorithms require different forms and extents of DPP. Thus, most relevant algorithms for production use cases are analyzed in the following. The choice of ML algorithms that are being discussed is made under consideration of the following criteria, which are formulated based on extensive literature research and supported by own project experiences at *Fraunhofer IPT*:

- Successful application in production (see Chapter 2.1)
- Tabular data (see Chapter 2.2)
- Supervised learning (see Chapter 2.3.2)
- Mix of baseline and more complex ML algorithms
- Mix of interpretable and non-interpretable ML algorithms
- Different levels of robustness against data quality issues

Figure 2-9 shows the overview of supervised ML algorithms with the assignment to the corresponding learning task of either classification or regression. In the following, each supervised ML algorithm is briefly introduced. Baseline models for both classification and regression range from *decision trees* (*DTs*), and *Gaussian processes* (*GPs*) over

k-nearest neighbors (*K-NNs*) to *support vector machines* (*SVMs*). For regression, *linear regression* and its regularized extensions of *ridge regression* and *elastic net* represent examples. In case of classification, *logistic regression*, *ridge classification* and *Gaussian Naïve Bayes* are discussed. Baseline models, also called weak learners, and *ANNs* can be ensembled in various forms illustrated by the arrow in Figure 2-9. The concepts of *bagging*, *boosting*, *stacking*, and *voting* as well as its exemplary ML algorithms will be outlined. Ultimately, the family of *ANNs* will be described.

Since the focus lies on recommending suitable DPP pipelines, the robustness against data set characteristics of every ML algorithm is outlined. ML algorithms can be robust according to the criteria of missing values, categorical features, outliers, unscaled, skewed, high dimensional, and imbalanced data. In addition the explainability and computational efficiency is investigated. For every criteria, it is checked if the robustness against a certain data set property, explainability, and computational efficiency is low or high. The results are subsumed in Table 2-1 and will be discussed in the following paragraphs.

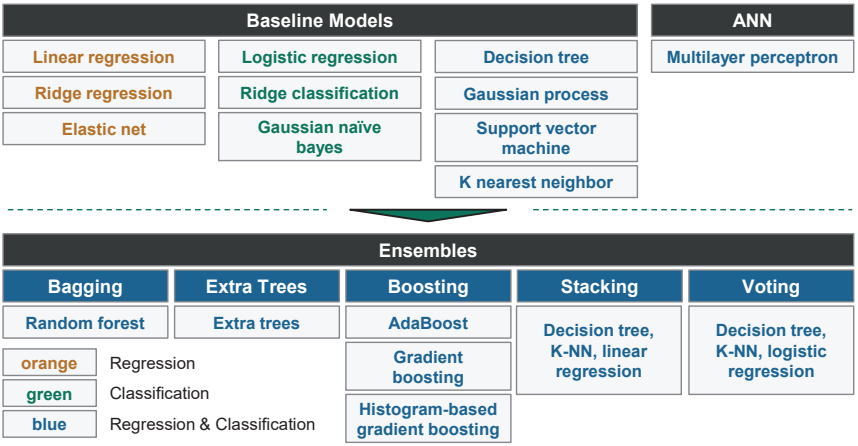


Figure 2-9: Supervised ML algorithms considered in this thesis

Linear regression

The principle of *linear regression* follows a linear combination of weighted attributes to determine a linear relationship between independent variables to a dependent variable. The application of *linear regression* requires the fulfillment of various assumptions. Samples need to be representative, while predictors should be linearly independent. Independent variables are free of errors, whereas errors of the dependent variable are uncorrelated. [YANG19, p. 70] The most basic *linear regression* model is based on

ordinary least squares, which aims at minimizing the sum of squared differences between actual values y and predicted values $\hat{y}$, so called residuals, based on good estimates β using gradient descent [GOOD16, p. 92]

Linear regression is extremely fast, and explainable [MOLN22]. However, in most production use cases, the assumptions of *linear regression* models such as independence of input features are not met. Further downsides are the susceptibility to overfitting for high dimensions. [FLAC12, pp. 195-197] Since ordinary least squares are also used to identify outliers due to its sensitivity towards outliers, when applying *linear regression* in the production context, outliers need to be handled accordingly [DABR20], [GARC15, p. 3]. In addition, *linear regression* cannot handle missing values properly and do not perform well in case of imbalanced data [BROW20d]. Features should be reduced due to being prone to overfitting. [YIUT19]

Logistic regression (classification)

In contrast to *linear regression*, *logistic regression* is used for two class classification. To map input features to a binary output, an interval [0, 1] is to be predicted by using a logistic function. Consequently, classes are assigned to the probability.

Besides the computational efficiency, *logistic regression* is easily interpretable and easy to implement [MOLN22]. The required DPP is comparable to *linear regression*. *Logistic regression* is highly sensitive to outliers and ineffective when making prediction in case of imbalanced data. High dimensional data impact the performance negatively, while scaling is not necessarily required since it shows comparable results to performance with scaled data [BOWN16]. As all ML algorithms, imbalanced data sets affect the final performance of the ML model [BROW20d].

Ridge regression & ridge classification

To overcome the drawbacks of ordinary least squares regression and avoid overfitting, regularization techniques are introduced. In general, regularization aims at reducing the error of the model by adding further information to the model. For *ridge regression*, by adding penalty terms to the error function, coefficients are shrunk by a uniform factor to become as small as possible, i. e., close to zero. *Ridge regression* and *classification* are easy to understand and computationally cheap ML algorithms. [BISH06, p. 10].

Compared to *logistic regression*, *ridge classification* exhibits lower computing times. Although being less prone to outliers, less dependent on correlated variables, preprocessing the data supports in achieving data sets with higher quality. Regularizations of ML algorithms generally lead to less DPP with respect to data reduction due to embedded feature selection [LIU08b, p. 405].

Elastic net (regression)

To combine the application of $l1$ and $l2$ regularization, *elastic nets* can be used. Since $l1$ regularization shrinks the coefficients to zero, too many variables can be removed from the data set. However, shrinking coefficients of sparse features to zero is desirable. On the contrary, shrinking coefficients close to zero can be helpful to reinforce the importance of different features. [FRIE10], [KIM07]

Thus, *elastic nets* are used in case of multiple highly correlated features, leading to embedded feature selection of the ML algorithm during training. Therefore, less preprocessing is required compared to *linear regression* [LIU08b, p. 405]. However, *elastic net* suffer from noisy data.

Gaussian Naïve Bayes (classification)

Another approach of mapping input data to class labels is to develop a probabilistic model. A conditional probability model aims at estimating the conditional probability of the class label based on the input features. The calculation of conditional probabilities can be conducted by applying the Bayes Theorem considering joint probabilities. [MURP12, pp. 29-30] By simplifying conditional probabilities to the assumption that input variables are independent from each other, the calculation of probabilities becomes manageable [MURP12, pp. 82-83]. This simplification is referred to as Naïve Bayes and assuming a Gaussian distribution.

For these reasons, Naïve Bayes requires little computing time and is explainable. Since Gaussian distribution is applied for Naïve Bayes classification, data needs to be encoded prior to the application. *Gaussian Naïve Bayes* (GNB) algorithms are prone to outliers [DABR20] and perform well on unskewed data. [KOLC05, pp. 561-565]

Gaussian process (classification & regression)

Gaussian processes (*GPs*) solve both classification and regression tasks and represent a generalization of the Gaussian probability distribution. Instead of describing independent variables as the Gaussian distribution does, *GP* considers the parameters of the functions. [RASM06, pp. 2-40, 79]

Due to the existence of different kernels, the performance of *GP* is generally high compared to other baseline models, while the computing time drastically increases in case of high dimensional data sets and explainability decreases compared to other baseline models. In relation to DPP, high dimensional data sets should be reduced since a high number of attributes also influences model performance. *Gaussian processes* require encoding prior to the ML application, and achieve better performance in case of independent variables being normally distributed. [MURP12, pp. 518-524]

K-nearest neighbors (classification & regression)

K-nearest neighbor (K-NN) is designed for both regression and classification. The goal of the distance based ML algorithm is to determine the target by considering the k closest neighbors. In case of regression, the prediction is the average of all *k-nearest neighbors* [SORJ05, p. 2]. In case of classification, the target class belongs to the majority of classes of considered *k-nearest neighbors*. To determine the target, a distance metric such as *Euclidean distance* is used [YANG19, p. 113], [GE17, p. 20603], [GÉRO18, p. 21]. *K-NNs* are widely applied due to its simplicity, however, are computationally expensive and not human-readable [WITT05, p. 78], [YIN15].

Referring to DPP, in case of high dimensional data sets, number of features should be reduced since distances grow as the dimensionality increases [MURP12, p. 18], [ZHUP14, p. 23]. The drawback of being less performant on high dimensional data sets is further amplified through the necessity of sorting learned data points by distance [YANG19, p. 112], [ZHUP14, p. 23]. Both outliers and unscaled data deteriorate the performance of *K-NN* due to its distance based metric [BHAA17, pp. 21-23], [DABR20], [BHAN20]. In addition, *K-NN* poorly performs on imbalanced data sets by highly preferring the majority class [GENE18].

Support vector machines (classification & regression)

Support vector machines (*SVMs*) were first developed by BOSER, VAPNIK, CORTES AND GUYON in several publications between 1992 and 1995 [BOSE92], [CORT95]. Originally, *SVMs* were designed for binary classification tasks. Later, modifications were made leading to the applicability of *SVMs* to regression tasks. [DRUC96] Due to its performance, applications such as condition monitoring in production can be found [WUES16, p. 38].

SVMs are hard to interpret and training times scale cubically [KOTS07, pp. 262-264]. Especially when using complex kernels, high computational resources are required [HAN12, p. 408], [WILL00, p. 1]. Referring to DPP, *SVMs* only work with numerical attributes, are sensitive to unscaled and imbalanced data [SOTE17], [WU03, p. 49], [SUN09, p. 694]. *SVMs* exhibit good performances in case of high dimensional data compared to aforementioned baseline models leading to less required data reduction [CHAK19]. They are further insensitive to outliers. [DABR20], [GARC15, p. 5]

Decision tree (classification & regression)

Another statistical approach being widely used in ML is the *decision tree* (*DT*). Nowadays, the commonly applied implementation is the classification and regression tree (CART) algorithm, introduced by BREIMAN ET AL. in 1984 [BREI84]. The CART algorithm divides the data set into sub sets in a binary manner depending on underlying

features. In case of classification, the leaf node represents a class, while in regression, the predicted value is the average of all instances within this leaf. [YANG19, p. 115], [BISH06, pp. 663-664]

As *DTs* scale linearly with the amount of features, computational efforts are relatively low and are interpretable [MOLN22]. In general, *decision trees* can handle missing values in the data, and are very robust against outliers since splitting the data does not rely on distance measures, but even penalizes high distances for regression tasks in case the MSE is applied [KUHN18, pp. 42, 174, 372], [ELG21], [DABR20], [JOHN95]. The influence of unscaled on the model performance is low, however, *DTs* tend to overfit in case of high dimensional data [BHAN20]. As most ML algorithms, *decision trees* are prone to imbalanced data [PEDR21e]. Due to the susceptibility to overfitting, *decision tree* are often referred to as weak learner. Weak learner can be ensembled to create more performant ML algorithms. Thus, in the following, the concepts of ensembles are introduced.

Ensemble learning

Ensemble learning aims at increasing the performance of an ML model by aggregating outputs of several ML models to one output [GÉRO18, p. 183]. By combining several weak or base learners, the bias-variance tradeoff is addressed (see Chapter 2.3.2) due to the significant reduction of base learners' variance, since the variance of one base learner is statistically mitigated by the large amount of other base learners [YANG19, p. 120], [MURP12, p. 550]. Ensembles are less prone to overfitting and obtain better generalization compared to base learners [HAN12, pp. 379-380]. Most common approaches of ensemble learning are *bagging*, *extremely randomized trees*, *boosting*, *stacking*, and *voting* that will be described in the following. While in *bagging* and *boosting*, base learners are homogeneous, i. e., one kind of ML algorithm, base learners in *stacking* and *voting* are heterogeneous. All tree-based algorithms are deemed explainable. [MOLN22]

Bagging - random forest (classification & regression)

Bootstrap aggregating, or short *bagging*, comprises the replication of the input data set through random sampling with replacement, while each resulting bootstrap maintains the number of instances of the input data set. [PHAM05, p. 10] Thus, each base model of the ensemble receives a slightly tweaked input data set, which is used for training the base learners resulting in different ML model outcomes. After training, outputs are aggregated. Among aforementioned advantages, RF exhibits a high computational efficiency, since every *decision tree* can be run in parallel [WU18, p. 5].

Extremely randomized trees (classification & regression)

Extremely randomized trees, short *extra trees*, are randomly fitted in parallel, but unlike *bagging* that performs random sampling, *extra trees* use the whole training data set [GEUR06]. Due to the minor changes compared to *bagging*, the robustness against data set properties are comparable with *decision trees*, however, *extra trees* tend to overfit on larger data sets [PEDR21d].

Boosting - AdaBoost (classification & regression)

In contrast to *bagging*, *boosting* algorithms do not use bootstraps from data to train base learners in parallel, but start with one base learner and subsequently train further base learners consecutively by considering results of previous learners [FRIE01]. The first *boosting* algorithm being capable of adapting to errors of weak learners is called *Adaptive Boosting* or *AdaBoost*, which is built upon *decision trees*. In case of incorrect model outputs, the instance receive increased weights, while correct outputs are assigned lower weights. [ROIG17, p. 414] *AdaBoost* is very prone to outliers, and imbalanced data [BROW20a].

Boosting - gradient boosting, XGBoost and LGBM (classification & regression)

Apart from *AdaBoost*, many *boosting* variations have been developed, mainly differing by the method how to weight incorrect outputs of the data and by applying different loss functions. One derivative of *boosting* is *gradient boosting* (*GB*). [FRIE01, p. 2] Inherently, *GB* scales linearly on the number of observations and features [FRIE01, pp. 36-37]. *GB* has proven the capability of achieving high performance in several use cases [KUMA20, pp. 55-57], [DANG17, pp. 169-171].

Other *boosting* variations range from *XGBoost* over *LightGBM* to *HistGradientBoosting*. *XGBoost* applies $l1$ and $l2$ regularization to improve model generalization and accelerates the splits by applying stochastic gradient descent. [CHEN16] To further accelerate the learning process, light gradient *boosting* machines (*Light GBM* or *LGBM*) were introduced. Due to the discretization of features, *LGBM* performs embedded feature selection [KE17]. Thus, compared to *GB*, *XGBoost*, *LGBM* requires less data reduction.

In general, all *boosting* algorithms weight misclassified data points during the training process accordingly resulting in more robustness to imbalanced data compared to other tree based algorithms. [CHEN21], [GARC15, pp. 118-120], [BATI04, pp. 20-22]. Due to the training procedure, outliers are also stressed during fitting the model, providing more robustness against outliers [SWAL18]. However, *boosting* algorithms tend to overfit on particular data sets by focusing too much on samples that are difficult to learn [VEZH07, p. 431], [LAUS08, p. 147], and suffer from high dimensionality [YIUT19].

Stacking (classification & regression)

Stacking combines multiple heterogeneous models to calculate an output. In contrast to *boosting* and *bagging*, where predefined metrics are used for aggregating individual model outputs, a meta model is trained in *stacking* to predict the final outcome [KUMA20, pp. 57-58]. Base learners can be any ML algorithm such as *decision trees*, *SVMs*, *K-NNs* or *ANNs* and the number of base learners can arbitrarily be set. The meta model is often a simple ML algorithm, typically linear or *logistic regression*, which aims at weighting predictions of base learners. [GÉRO18, pp. 200-202] The choice of baseline models influences the necessary DPP.

Voting (classification & regression)

Voting ensembles consider the output of multiple ML models that are trained in parallel and can be applied for both classification and regression. Baseline models should perform comparably well on the task to achieve overall good estimates. The necessary DPP and the robustness against data properties are dependent on the baseline learner being chosen.

Artificial neural networks - multilayer perceptron (classification & regression)

Artificial neural networks (*ANN*) comprise computational models inspired by the functionality of a human brain [MITC97, p. 82]. *ANNs* are applicable for both classification and regression as well as any linear or nonlinear function [GE17, p. 20601]. Every *ANN* consists of n input neurons, called input layer, and m output neurons, called output layer. There is a variable amount of hidden layers between output and input layer. For an artificial neuron, inputs x_i are received and forwarded by connections with corresponding weights w_i. After summing all inputs and adding a bias b to it, an activation function $\sigma(\cdot)$ is applied to incorporate nonlinearity to the function, as stated in Equation (2-6). [AWAD15, pp. 129-130], [GOOD16, pp. 171-174]

$$y = \sigma\left(\sum_i w_i x_i + b\right) \tag{2-6}$$

Arranging neurons of a network in a specified manner forms the architecture of *ANNs*. A basic structure of an *ANN* is a fully connected feedforward neural network, called *multilayer perceptron* (*MLP*), in which a variable number of hidden layers exist between input and output layer. [YANG19, pp. 140-141], [MITC97, p. 83] Figure 2-10 shows an example of an *MLP* with four layers.

In general, *ANNs* are widely used due to their applicability and suitability to handle complex tasks. Hurdles of using *ANNs* are the complexity of the algorithm' architecture,

the long training times as well as the susceptibility to overfitting through the generation of highly complex models [WUES16, pp. 36-37].

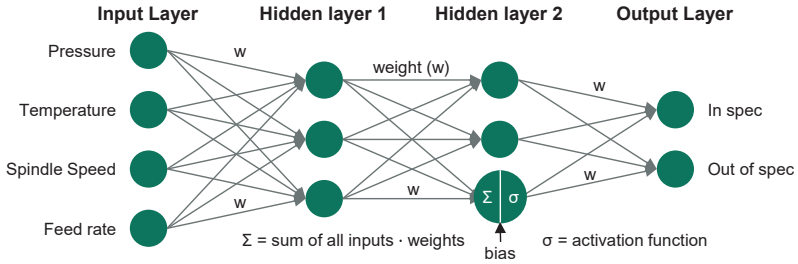


Figure 2-10: Multilayer perceptron with four layers

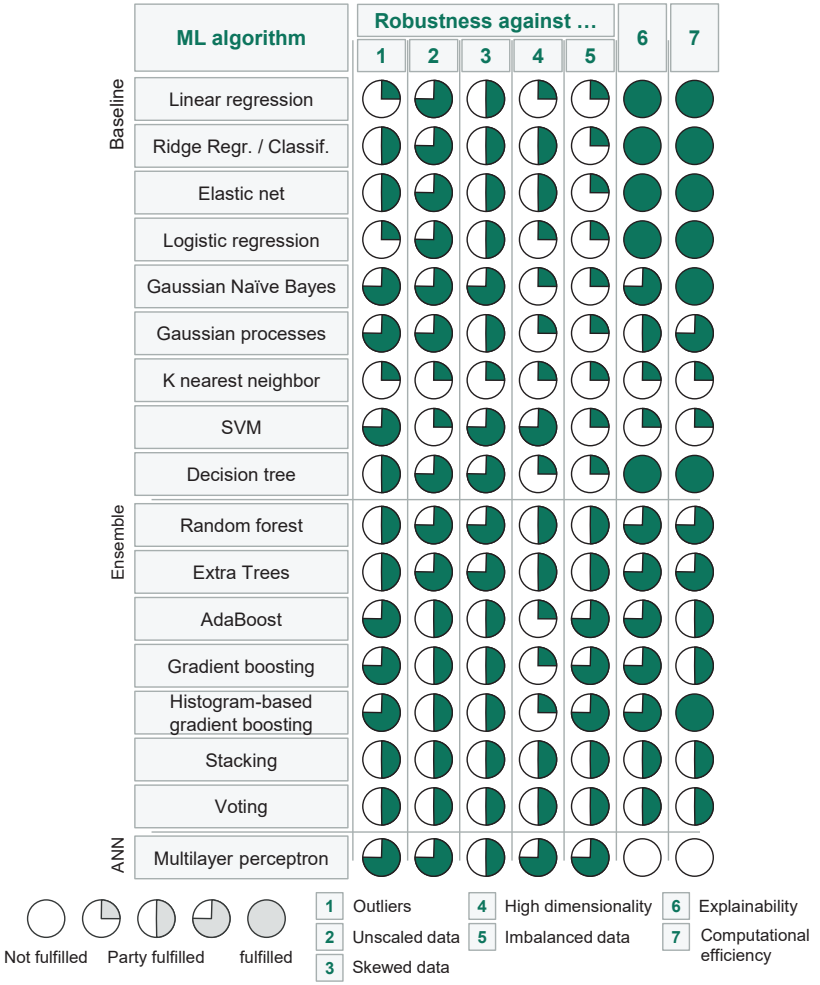
ANNs can only work with numerical data making data encoding essential prior to fitting an *ANN* [GARC15, pp. 3-5]. Since *ANNs* require a lot of data, augmentation and oversampling strategies can be followed to increase the number of instances. Also, since many *ANNs* such as *MLP* assume normally distributed features, corresponding transformers should handle the skewness during DPP. Scaling improves the performance of *MLPs* [BROW19b], while *MLP* can handle imbalanced data sets, however, the performance can be enhanced with DPP [KIBE20]. *MLPs* further benefit from reducing data dimensions [SING20, pp. 1-3].

Interim conclusion

Table 2-1 summarizes the findings about the robustness of ML algorithms against different data set properties. Harvey balls are used to express to what extent individual ML algorithms are robust against corresponding data set characteristics. Considered characteristics range from outliers and unscaled data over high dimensional to imbalanced data. Since being deemed important in production, the findings for explainability and computational complexity are shown.

In ML projects, python is commonly used as programming language due to having a large community with high popularity, and offering high flexibility in model building. In addition, sklearn is the most often applied python library when creating ML models presented above [CLAR18]. One exception represents the application of *LGBM*, which is used from the LightGBM library [MICR22]. Although by nature, some ML algorithms can cope with the presence of missing values, or categorical features, sklearn neither support missing values nor categorical features in the data set making it essential to preprocess the data accordingly [KUHN18, pp. 42, 174, 372], [PEDR21e], [PEDR21b], [PEDR21d], [PEDR21c], [PEDR21g].

Table 2-1: Overview of robustness of ML algorithms against data quality issues



Since sklearn is used for recommending DPP pipelines in this work, the focus of Table 2-1 lies on the robustness against data set properties given the sklearn library as basis. When using sklearn, missing values and categorical features always need to be pre-processed, which is why they are not depicted in Table 2-1. Harvey balls are entirely filled if a certain algorithm is highly robust against a data quality issue, explainable or exhibits low computational efforts. Table 2-1 emphasizes that ML algorithms cope with data set characteristics very differently. Consequently, DPP is essential for the success of ML projects, which is discussed in detail in Chapter 2.4.

2.4 Data Preprocessing

Data preprocessing involves the process of applying all necessary techniques to transform raw data in a high-quality data set [GARC15, p. 11]. Since it is such a crucial task for any data driven analysis, the mandatory provision of high data quality is outlined in the first sub chapter. Subsequently, characteristics of production data are discussed and criteria that influence DPP examined (Chapter 2.4.2). Since there are hundreds of methods for preprocessing tabular data, an overview and categorization are provided on DPP methods in Chapter 2.4.3. Subsequently, design rules for creating suitable DPP pipelines are outlined (Chapter 2.4.4).

2.4.1 Necessity of Data Preprocessing

Since data quality is such a crucial factor for data driven projects, standards have been developed to standardize its assessment. ISO/IEC 25012 aims at identifying the degree of fulfillment of data quality by defining 15 characteristics, which are categorized in inherent, system-dependent, or in-between [ISO08]. The general recognition of the importance of data quality is also reflected in the variety of taxonomies that have evolved to identify data quality [WANG96], [REDM96], [APEL15]. To address the system dependent quality characteristics of availability, portability, and recoverability, a process-oriented approach needs to be followed requiring the analysis and adaption of the IT infrastructure [WANG96], which is left out of consideration in this work. The inherent data quality characteristics of accuracy, completeness, consistency, credibility, and currentness can be addressed through a data-oriented approach. This work follows the data-oriented approach that involves DPP. Although providing a concept to get a transparency of company's data, a more thoroughly approach needs to be followed for determining the data quality for ML applications. [SARF20]

In ML, poor data quality has the greatest impact on the quality of the output [REDM18]. According to KOTSIANTIS ET AL., data quality is first and foremost for every ML endeavors [KOTS06]. First, preprocessing the data ensures technical prerequisites for applying ML, i. e., DPP is an enabler for any data driven analysis. Real world raw data is commonly incomplete and inconsistent due to the occurrence of missing values, different data types or high dimensionality as mentioned in Chapter 2.1 [GARC15, pp. 59, 148-149]. As shown in Chapter 2.3.3, ML algorithms require input data in specific formats.

Preprocessing the data is thereby the key success factor for achieving high quality. DPP has *essential impact on data quality and, hence, ML model performance*. Poor data quality certainly leads to poor performance of ML models. This phenomenon is commonly stated as "garbage in, garbage out", so called GIGO [PYLE99, p. 26]. Even

if an ML algorithm is able to generate results on a raw data set, results can be inaccurate or illogical [GARC15, p. 11]. DPP improves both accuracy of ML models and efficiency of ML model training. For instance, data set characteristics can be changed for performance increase or the data set's size can be reduced to decrease training times. [GARC15, pp. 39-40] GIOVANELLI ET AL. have proven that applying DPP methods in a sequence increases the ML model performance. [GIOV21, pp. 1-3]

In conclusion, data quality is the key factor for data driven decision making and ML. This work focuses on inherent data quality characteristics and data-oriented improvement of the quality. DPP is thereby a crucial step towards generating performant estimators. For production, further characteristics and requirements apply, which will be introduced in the following to be able to identify most suitable DPP methods.

2.4.2 Production Data Characteristics and Requirements on DPP

In production, data is obtained from different sources and can have varying forms and characteristics [GILL21]. Sources include machine tools, processes, quality inspection, or manual inputs. Data can come also from different production systems such as ERP, MES or SCADA systems, referred to as automation pyramid [KIES19, p. 22]. To gain a better understanding of which characteristics need to be addressed during DPP, the properties of data format, and specific data set properties for tabular data are outlined in the following. The characteristics are used to define meta features for describing given use cases.

Data format

Production data comes in different formats and can be divided into text, image, audio, video and database-based data. Each data format can assume different file formats. [UNIV21] A short overview of common file formats is provided.

- *Text* can be presented in the form of books, emails, or documentation. Text based data is often used in XML, PDF, HTML or plain text formats [GUPT17], [UNIV21].
- Typical *image data* formats are TIFF, PNG, JPEG, BMP, GIF [UNIV21]. In microscopy, further image formats exist such as nd2 or czi.
- Digitally recorded sound is called *audio* and is also increasingly used in data analysis. [SHIN20, p. 1] Popular file formats of audio data are WAVE, AIFF, MP3, MXF and FLAC formats [GUPT17].
- In *video*, images and audio streams are combined with each other. Exemplary file formats are OV, MPEG-4, AVI, MXF formats [UNIV21].
- The most common data format used for data analysis is *tabular data* [GODF16, p. 2143]. Vast majority of ML algorithms require vector-based numerical data.

For this, tabular data is particularly suitable since it can be easily split into vectors. Tabular data can be found in most databases. Common file formats are CSV, TDMS, TAB or SQL. [GUPT17], [UNIV21]

Tabular data set characteristics

Due to its increasing generation in production, the work focuses on tabular data. In the beginning, terms for tabular data are defined. Figure 2-11 shows the naming convention of tabular data sets. To represent the dimensionalities of the data frame, meaning number of rows and attributes, the shape of the data is investigated. While rows can also be named as instances or observations, columns can also be stated as dimensions, attributes, or variables. Features are interchangeably used with attributes, while features are sometimes referred to the final set of attributes that is used for modelling.

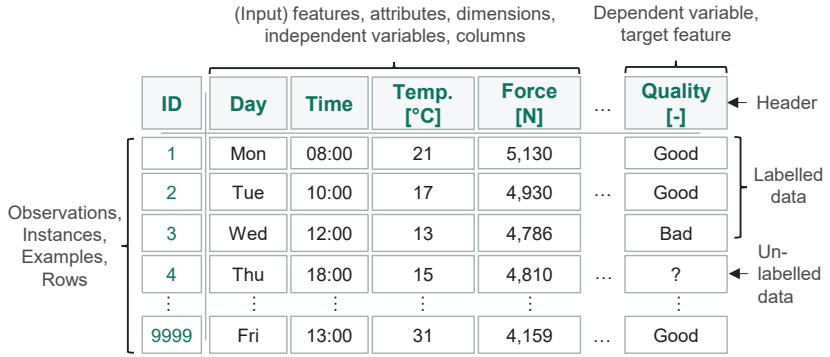


Figure 2-11: Naming convention for tabular data sets

In general, data sets comprise an identifier (ID), and header. The target variable is also called dependent variable. Data types, inner data structure, and the target variable of tabular data will be described in more detail in the following.

Tabular data set characteristics – data type

Data types can be classified as either numerical or categorical and influence, whether and how ML algorithms can read and process this underlying data [PYLE99, p. 53]. Each data type is introduced including its data scales in the following, which are subsumed in Figure 2-12.

- *Categorical data,* or qualitative data, represents categories and can be further classified into nominal and ordinal data. Nominal data represents classes of distinguishable objects and have no direct relation with each other. For instance, there is no order of different production sites of a company such as Aachen, Florianopolis, or Cuernavaca. In case of ordinal data, classes have a ranking

and relationship. An example is the tool wear, which can be classified as high, medium, or low. A special case of categorical data is the Boolean data type, where the number of possible categories degenerates into two, often represented as true and false or 1 and 0. The assessment of product quality as stated in Chapter 2.1, whether it is in specification or out of specification may be one example for a Boolean data type. [PYLE99, pp. 53-55]

- *Numerical data,* or quantitative data, are numbers of an infinite range and can further be divided into interval and ratio scale. In case of interval, numbers do not have a natural 0 value. An example is the temperature in [°C], which can also assume negative values. Numerical values with a natural 0 value are assigned to ratio. The spindle speed in [rpm] is one example for ratio scale. [PYLE99, pp. 55-58]

	Data Scale	Definition	Production Examples
Categorical	Nominal	Classes of **distinguishable objects**	Material: Aluminum, Titanium
Categorical	Ordinal	Classes with a **rank order relationship**	Tool wear: Low, Medium, High
Numerical	Interval	Numbers **without a natural 0** value	Date: 2020-05-07 Temperature: - 5.0 °C
Numerical	Ratio	Number with a **natural 0 values** and unit	Weight: 2,176.7 kg Temperature: 270,4 K

Figure 2-12: Definition and production examples of categorical and numerical data

Besides numerical and categorical data types, further special cases are defined to cover more specific data set characteristics. Another variant of data type is the date, which stores a timestamp and can contain a date and time. Thereby, data points are chronologically related to each other.

Tabular data set characteristics – inner data structure

The inner data structure can be distinguished in *time-series, cross-sectional,* or *panel data*. *Time series data* represents a collection of instances for a single item at different time intervals. For instance, data can be acquired over time in a milling process only for one specific target variable, e. g., accelerations during machining. [BROC02, pp. 1-7] *Cross-sectional data* is an accumulation of instances for multiple items at a single point in time. Given the example from Chapter 2.1, another milling process at the same time with different machine temperatures also acquires accelerations during machining in that point in time. [SIEG12, p. 429] *Panel data* is a combination of time-series and cross-sectional data by representing an accumulation of instances for multiple items at

multiple points in time. [BUTE20] Inner data structures focused in this work are *cross-sectional*, *time series data* as well as *panel data*.

Tabular data set characteristics – target variable

The target variable can assume different data types and is dependent on the learning task. In case of *classification*, the label consists of categories, either ordinal or nominal. In terms of data quality, the representation, i. e., number of instances per class, needs to be determined. If a *regression* task is present, the target variable is numerical, meaning either in a ratio or interval scale. In order to identify the label's quality, skewness and kurtosis is calculated [BLAN13].

Data preprocessing requirements

Production data characteristics give insights about its quality, while placing versatile requirements on DPP, and being highly use case specific. By nature, requirements on DPP are not solely derived from raw data but are also influenced by robustness of ML algorithms against data set characteristics (see Chapter 2.3.3). In total, data set characteristics, ML algorithms, production requirements and DPP itself build four segments of influencing factors on DPP, which can be seen in Figure 2-13, and which are introduced in detail in the following. [FRYE20, pp. 243-245]

Data preprocessing requirements – data set characteristics

Data preprocessing is highly influenced by data set properties, which can be very different over data sets. One basic property is the shape, which shows the number of instances and features of a given data set. If the number of instances is too low, ML algorithms may simply not converge during training or may not generalize well due to the existence of too less noise. A data set with too many instances may have too much noise. Then, selected ML algorithms only learn noise instead of memorizing key features in the data. Having too less features in the data set may lead to a too loose relationship between independent and dependent variables, while too many features can exhibit highly correlated features and bias the ML algorithm. Therefore, DPP needs to handle data set properties to ensure the correct level of noise. These design decisions are directly linked with the bias variance tradeoff introduced in Chapter 2.3.2.

Typical data quality issues such as missing values, outliers, and the number of duplicates, or constant features need to be handled. Referring to the data type, nominal and ordinal features pose different requirements on DPP, since an ordinal structure of features should remain. Different data ranges and skewed data affect DPP as well as high dimensional, or imbalanced data.

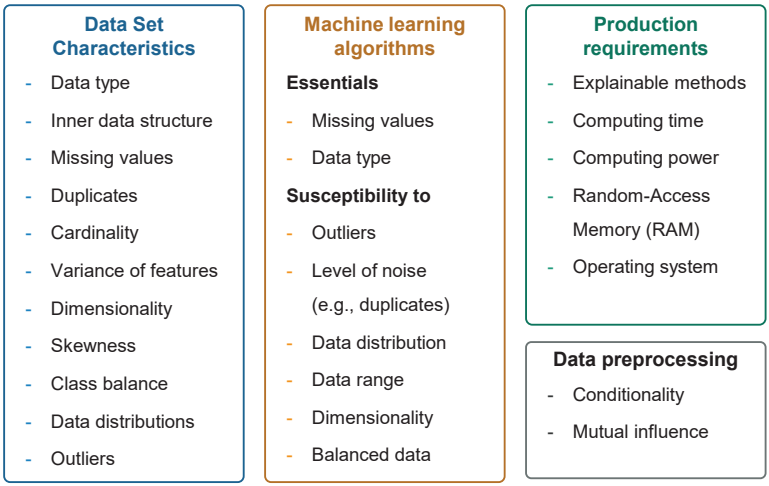


Figure 2-13: Four segments that influence DPP

Data set characteristics can be determined through the extraction of meta features. Dependent on the characteristic, the meta feature acquisition can be easy, e. g., in case of retrieving the shape, or costly, e. g., in case of calculating the distributions or outliers for every input feature. [VANS19, p. 8] Strategies on how to address data quality issues through DPP is examined in Chapter 2.4.3.

Data preprocessing requirements – ML algorithms

ML algorithms in its common implementations require a complete data set without missing values and solely numerical features, highlighted as essentials in Figure 2-13. Dependent on the ML algorithm at hand, the lacking robustness against certain data set characteristics deteriorates the performance as found in Chapter 2.3.3. Therefore, an emphasis should be put on outliers, level of noise through constant columns or duplicated rows, as well as ranges and distributions of data [BHAA17, pp. 21-23], [DABR20]. Many algorithms are prone to overfitting in case of high dimensional data and do not cope well with imbalanced data, making DPP important for quality leaps [WU03, p. 49], [SUN09, p. 694], [YIUT19].

Data preprocessing requirements – production requirements

The user in production may require explainable DPP methods and ML algorithms to be used to be capable of interpreting and understanding the outputs [BELL21, pp. 1-10]. Based on the interpretability of DPP methods and ML algorithms, production experts can influence the creation of the ML pipeline and further optimize the outputs of ML models [MOLN22]. Computing resources, i. e., the computing power, either CPU,

or GPU, and random-access memory (RAM) may be limited in production. Thus, DPP methods need to be computationally cheap and computing times of DPP methods need to stay within a manageable range [KLIM21]. Here, the operating system (OS), e. g., Windows or Linux, also can have an influence on DPP methods to be used, since some methods are tailored for a corresponding OS.

Data preprocessing requirements – data preprocessing

When selecting and combining the large number of existing DPP methods into DPP pipelines, conditionality and mutual influence of DPP methods need to be considered. As ML algorithms, DPP methods require specific data properties to work properly. For instance, the application of a *principal component analysis* (*PCA*) involves a prior data scaling to manage different ranges of features [FANC21, p. 7]. Furthermore, different DPP methods impact each other, which can lead to DPP methods not being executed as intended. In case a *low variance filter* is used, dependent on the threshold based on which a feature is removed, a prior application of *MinMaxScaler* can result in the execution of *low variance filter* without any impact since the *MinMaxScaler* influences the variance of the feature.

These requirements shown in the four different segments need to be met when preprocessing tabular production data. In conclusion, DPP need to:

- handle different data set characteristics such as missing values,
- consider susceptibility of ML algorithms to data set properties such as being prone to imbalanced data by choosing corresponding methods to address the problem,
- comply with external requirements, e. g., selecting solely explainable DPP methods for production use cases, and
- involve a set of mutually influencing DPP methods to achieve higher ML model performances.

To meet the requirements that are placed on DPP, many different methods have been introduced in recent years. For that reason, Chapter 2.4.3 structures data preprocessing and provides an overview of available DPP methods.

2.4.3 Data Preprocessing Methods

In the following, a structured overview about DPP methods is presented, which form the basis of DPP pipelines. Versatile taxonomies of DPP have been developed. GARCIA ET AL. developed an overview of DPP methods that are classified into data cleaning, data transformation, data reduction, and data integration [GARC15, pp. 11-16]. BROWNLEE distinguishes between data cleaning, feature selection, data transformation and dimensionality reduction, while HAN ET AL. divides DPP into cleaning, integration,

reduction as well as transformation and discretization [BROW20c], [HAN12]. FAN ET AL. proposes data cleaning, data transformation, data reduction, and data scaling as DPP steps [FANC21]. ABDALLAH ET AL. additionally covers data sourcing and integration prior to data cleaning, data transformation and data reduction [ABDA17]. In conclusion, the presented approaches cover many steps of DPP but do not fully represent the steps to be performed in the production domain, especially in presence of imbalanced data, while lacking a thoroughly and detailed assignment of DPP methods to each DPP step.

FRYE & SCHMITT developed an approach considering the production domain, in which the steps are classified into data integration & synchronization, data cleaning, data transformation, data reduction and data augmentation & balancing (see Figure 2-14). Due to representing an highly iterative task while being dependent on the IT experts and according to Figure 1-1 as well as explanations in Chapter 2.2, techniques that deal with integrating and synchronizing of data are not considered. [FRYE20]

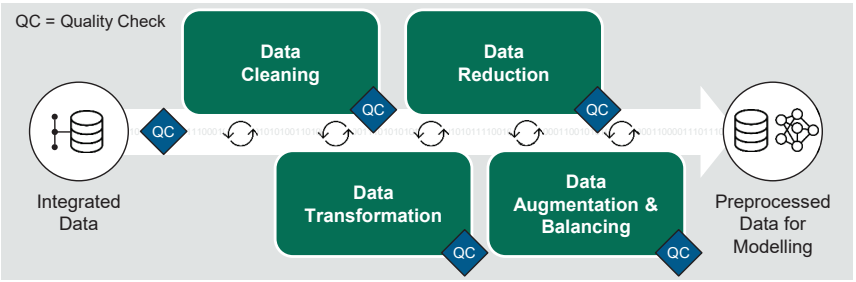


Figure 2-14: DPP steps adapted from [FRYE20]

Based on an integrated and synchronized data set, a data quality check is initially performed. Subsequently, data is cleaned by handling missing values, outliers, and noisy data. During transformation, data can be encoded, scaled, and discretized. Skewness handling also represents one common step when transforming data. This is followed by reducing the data set's shape by applying techniques from dimensionality reduction or feature selection, and instance selection. In the next step, data is augmented to enlarge data set size by adding further instances to the data, including balancing for both classification and regression tasks. The result of DPP is a preprocessed data set ready for modelling. In between, data quality is assessed through the application of quality checks.

In practice, preprocessing data represents an iterative approach, in which the order of methods can change dependent on the given use case, which is illustrated by the rotating arrows in Figure 2-14. Each step will be described in the following by dividing it

into subsections and depicting its purpose. Thereby, different categories are introduced per DPP step. Eventually, existing and relevant DPP methods are listed and assigned to the DPP category. For sake of a transparent overview, a long list for DPP methods has been created according to the corresponding DPP category, which can be taken from Annex A.1-A.4. An intensive research has brought more than 200 commonly used DPP methods. Since new methods are continuously being developed, existing concepts transferred to new applications, the author makes no claim to completeness.

Data cleaning

Data cleaning handles missing values (MVs), outliers, and noise in the data set (see Figure 2-15), which are outlined in the following the subsections. [GARC15, pp. 11-13] The application of such DPP methods is necessary in production, for example, for sensor or machine data. [PHAM05, p. 405], [SYAF18], [WUES16, p. 26].

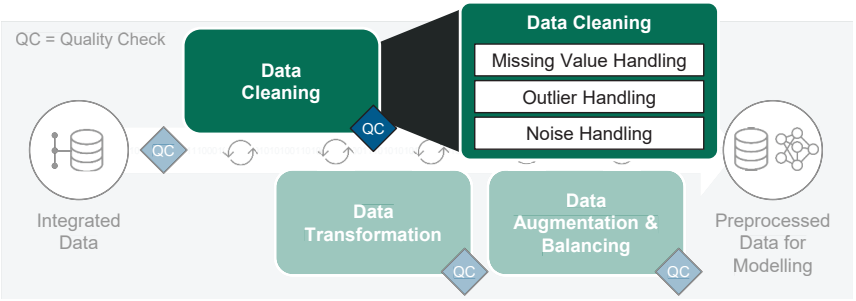


Figure 2-15: DPP categories of data cleaning

Data cleaning – missing value handling

Missing values are common in production due to artefacts from incorrect measurements, sensor defects or human failure [GARC15, p. 59]. According to Chapter 2.1, an example for incorrect measurements is the overloading of a machine tool during a cutting process, which receives too much information for re-adjusting its path. This event can result in MVs since the machine controller is prioritizing the readjustment instead of writing information to the machine console. Human failure can occur fraudulent or randomly by mistake. Challenges arising from MVs are generating bias, efficiency decrease, and problems in case ML algorithms are not capable of handling MVs. [BARN99, p. 17]

In theory, MVs can be distinguished in three types. In case values are *missing completely at random* (MCAR), no reasoning can be derived from the MV's presence. For instance, spindle speed, drive load and temperature are acquired during machining. If

missing values are MCAR, MVs in the feature spindle speed are not related to drive load or temperature. Tests for identifying MCAR are *Little's MCAR test* or *logistic regression*. [LITT88, p. 1198], [SCHE02, p. 154] *Missing at random* (MAR) values depend on other feature values, i. e., the reason for the existence of a MV lies in values of other features. The occurrence of a MV in the spindle speed can be too high temperatures, which is why the sensor failed to acquire data. *Missing not at random* (MNAR) applies if the cause that results in the MV is the attribute at hand. The temperature sensor fails due to too high temperatures. [SWAL18] Determining the occurrence of MAR and MNAR is challenging in practice. However, the treatment of MAR, MNAR and MCAR requires different types of preprocessing methods, while every DPP method introduces bias to some extent. As Chapter 2.3.3 reveals, ML algorithms used in common libraries cannot handle MVs. To preprocess data respectively to their appearance, MVs can be handled by ignoring, deleting or imputing. Figure 2-16 shows an excerpt of DPP methods according to their type of MV handling [FARH07, p. 692], [GARC15, pp. 63-64].

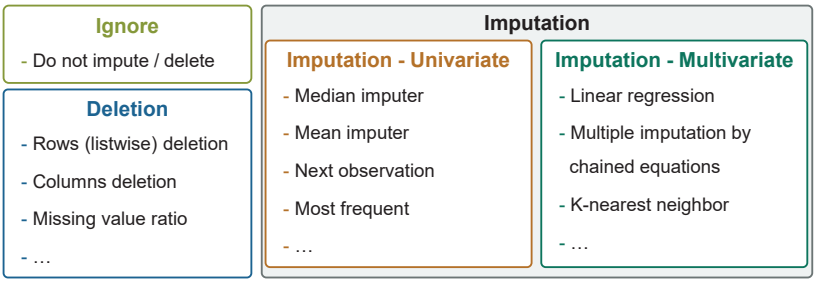


Figure 2-16: Classification of missing value handling and an excerpt of techniques

In case MCAR values are predominant in the data set, missing values can be ignored if the number of MVs is low and ML algorithm handles missing values. MCAR values can be deleted if many MVs are available. [SCHE02, p. 160]. By deleting entire rows or columns from the data set, MVs can be deleted. If MVs are spread over many columns, data is imputed. Univariate imputation, e. g., *MedianImputer* that imputes MVs based on the column's median, addresses MCAR by taking into account only one column at hand. MAR values are imputed based on multivariate imputation approaches such as *K-NN imputer* through consideration of different columns in parallel. Lastly, MNAR is hardly to detect in practice and needs to be avoided since it introduces irreversible bias in the data set [SCHE02, p. 154]. Handling MVs starts with the application of simple imputing techniques and subsequent increase of methods' complexity. A list of 31 DPP methods for MV handling including comprehensive descriptions can be taken from Annex A.1.

Data cleaning – outlier handling

Outliers represent extreme values in a data set differing from surrounding data points that stem from natural variabilities, human failures as well as errors during the recording of data via sensors. [SANT17] As for MVs, outliers can be of three different types: *global*, *contextual*, or *collective* [HAN12, p. 545]. The treatment of outliers thereby depends again on its corresponding type [HAN12, p. 548]. Global outliers are single data points exhibiting high distances to the rest of the distribution of the data set. Data sets contain contextual outliers if a data point is different from the pattern of the distribution. Collective outliers further appear in case a sub group of data points differ from the data set. Once detected, analogous to the treatment of missing values, outliers can be either ignored, deleted, or imputed [WILS16], [MAGA21]. As shown in Figure 2-17, outliers can be identified in an univariate or multivariate way.

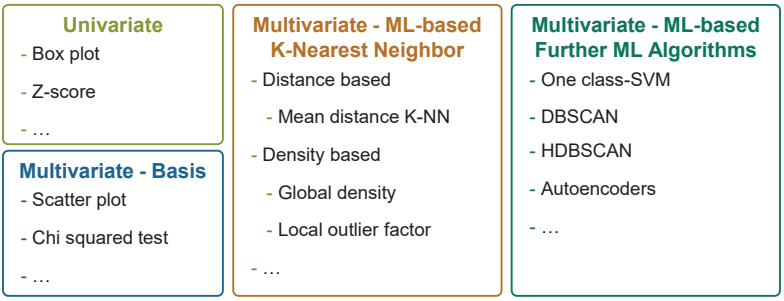


Figure 2-17: Classification of outlier detection and an excerpt of techniques

In practice, global outliers can be identified through simple, univariate methods such as *z score* or *box plot* [HAN12, pp. 554-555]. Detecting contextual outliers require multivariate methods, starting from basic *scatter plots* up to methods that stem from ML. For multivariate methods, a distinction can thus be made between basis, *K-NN* and further ML based detection techniques. Prominent examples are *local outlier factor* or *one class-SVMs* [HAN12, pp. 571-573]. Collective outliers are very hard to identify. A long list of 17 DPP methods for outlier detection including descriptions can be seen in Annex A.1.

Data cleaning – noisy data handling

The presence of noise in data is a common problem that produces negative consequences in data analysis. Noisy data represents data with large amounts of additional and meaningless information. [GARC15, p. 13] On the contrary, the absence of noise result in overfitting when training ML algorithms [GARC15, p. 22]. Therefore, noise contributes directly to the bias variance tradeoff (see Chapter 2.3.2). Noise is further in inevitable problem that comes from natural deviations in measurements, human

faults, or sensor errors [GARC15, pp. 114-115]. Three types of noise have been introduced, which are *instance*, *attribute*, and *class noise* and can be seen in Figure 2-18 including relevant methods [LUEN21, pp. 390-391].

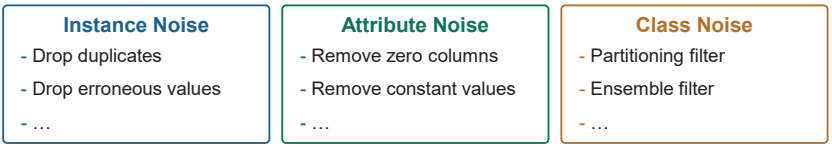


Figure 2-18: Classification of noisy data handling and an excerpt of techniques

Instance noise can exemplarily contain duplicated values that are addressed by *data deduplication*. In addition to the items of instance noise, attributes can have constant values as well as feature with zero values that can be removed from the data set. Class noise appears in case of classification tasks if instances are mislabeled or contradictory. [GARC15, pp. 114-115] For addressing class noise, noise filters have been developed to eliminate noisy instances in the data set. Examples are *partitioning* or *ensemble filters*. A long list of 16 DPP methods for noisy data handling are found in Annex A.1.

Data transformation

As already discussed in Chapter 2.4.2, data can come in different data types but also different ranges. For handling categorical features, data encoding is applied, while scaling is performed to address different ranges of features. For instance, spindle speeds may range between 0 and 12,000 mm/min, while ambient temperature values are moving from - 10 °C to 30 °C. Since ML algorithms may interpret higher values as more important, data is scaled. [GARC15, pp. 46-48], [ABDA17] Data can also come in different distributions such as binomial or multimodal. However, ML algorithms often show better performance in case underlying data is normally distributed [HAN12, p. 556]. Therefore, distribution's skewness is handled. Lastly, numerical columns may have a high number of different values within one column, i. e., cardinality, leading to the potential to introduce too much noise in the data set. By data discretization, numerical data can be categorized. In the following, the DPP categories of data transformation are presented, namely, encoding of categorical values, scaling, skewness handling, and discretization as shown in Figure 2-19.

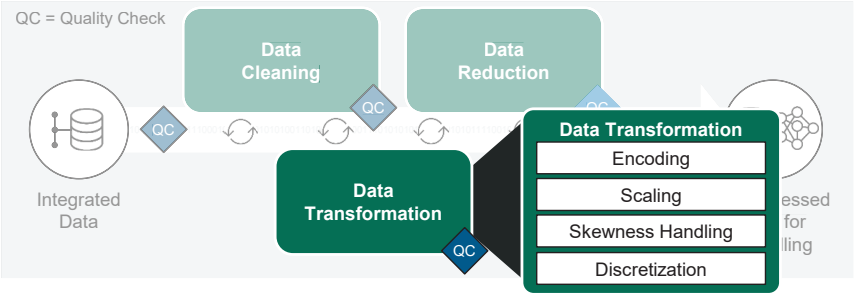


Figure 2-19: DPP categories of data transformation

Data transformation – data encoding

The presence of categorical features can pose problems for ML algorithms, as they often cannot process categorical values correctly. Categorical encoding is applied to convert categorical into numerical data. Different encoding methods express different relations between categories [FANC21, pp. 8-9]. The goal of encoding is to select the most appropriate encoder to preserve the original meaning of the categories. Categorical encoders can be divided into classic, Bayesian, and contrast encoders. [HALE18] While *classic encoders* are simple, easy to understand and very popular, *Bayesian encoders* use information of the target variable in combination with categorical variables. *Contrast encoders* consider mathematical patterns among categories. [CARE03, pp. 10-11] The overview of different encoder types and an excerpt of corresponding methods can be taken from Figure 2-20.

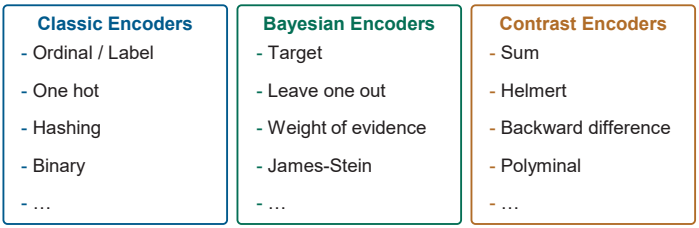


Figure 2-20: Classification of data encoding and an excerpt of techniques

Different types of categorical data (nominal/ordinal) require different encoding methods. For nominal categorical data, classical encoders such as *OneHotEncoding*, *Hashing*, or Bayesian encoders like *leave one out*, and *target* can be applied. In case of high cardinality and decision tree-based algorithms, the application of *OneHotEncoding* should be avoided [FANC21, p. 9]. In case ordinal data is present, classical encoders such as *LabelEncoder*, or *TargetEncoder* can be applied. Contrast encoders, e. g., *Sum*, *Helmert*, *Backward Difference*, *polynomial encoders* have yet not often been

used in production [BUIH20], [UCLA21]. Annex A.2 provides a long list of 20 DPP methods for data encoding including comprehensive descriptions of techniques with advantages and disadvantages.

Data transformation – feature scaling & handling skewness

Since features can contain different ranges through unit representations such as the consideration of pressure in units of *Pa* or *bar*, and magnitude representations when comparing different features with each other, e. g., spindle vibrations and flank wear [NASA21], feature scaling is performed. The application of feature scaling techniques aims at dissolving the dependence on measurement units and different magnitudes of real data [HAN12, p. 99]. Transforming features from actual to a standard range of values generally improves performance of ML models. For example, different feature scales in distance-based ML algorithms, such as the *K-NN* algorithm, lead to a different weighted evaluation of the features by the algorithm, which directly affects the performance. [BHAN20] Therefore, scaling is used so that all features contribute equally to the result. An overview of scaling techniques is given on the left of Figure 2-21, while *standardization* and *rescaling*, sometimes also referred to as *normalization*, represent most common methods for scaling [BHAN20], [ROYB20], [CHON20].

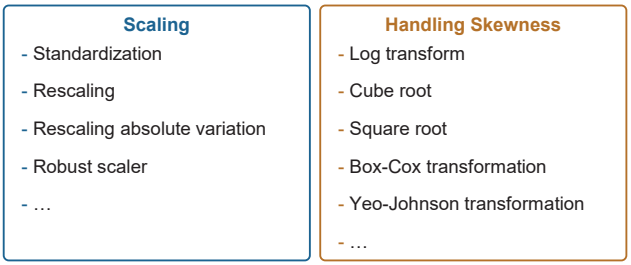


Figure 2-21: Excerpt of techniques for scaling and handling skewness

Skewness is the measure of asymmetry of a probability distribution. Skewed data can exhibit any common distribution shape. Since ML algorithms prefer data in normal distribution [HAN12, p. 556], so called power transformers are used to remove skewness [GARC15, p. 54]. On the right of Figure 2-21, a selection of methods for handling skewness is shown. In case data is positively or right-skewed, applicable power transformers are *square root*, *cube root* and *log transform* [HAN12, p. 106]. *Square root* is only applicable to positive values, while having only a small effect on the data distribution. *Cube root* can be used both for negative and positive values and have a moderate effect. The biggest effect on skewed data has *log transformations*. [HAN12a, p. 106] If negatively or left-skewed data appears, *cube root* and *log transformations* can be applied as well. However, instead of using square root, only square transformation is

applied. [BROW20e] More performant power transformers are *Box-Cox* and *Yeo-Johnson* power transformers. While *Box-Cox* is only applicable to positively skewed data, *Yeo-Johnson* can also be used in case of negative skewness [KUHN20, pp. 122-124] In Annex A.2, a list and thoroughly descriptions of ten power transformers as well as scaling methods can be found.

Data transformation – data discretization

Numerical attributes can have very high cardinality, while a high number of instances can be combined without losing too much information. The goal of discretization is mapping huge spectra of numeric values to a reduced sub set of discrete or nominal values by establishing an association between each non-overlapping interval with a numerical discrete value. Data discretization can also be stated as data reduction technique. [GARC15, p. 15] The process of discretizing data is described in four consecutive steps. In the beginning, an ordering of continuous values for a respective attribute is performed that can either be descending or ascending. An optimal cut point or pair of adjacent intervals is determined to either split or merge feature values. The second step comprises the correlations calculation by applying an assessment measure. Thirdly, the entire data set is either merged to adjacent intervals or split into partitions. The process of discretization ends if a stopping criterion is reached. [GARC15, pp. 247-250] Discretization methods can be divided into binning, statistical or information based (see Figure 2-22). A more thorough overview can be taken from GARCIA ET AL. [GARC15, pp. 254-258]

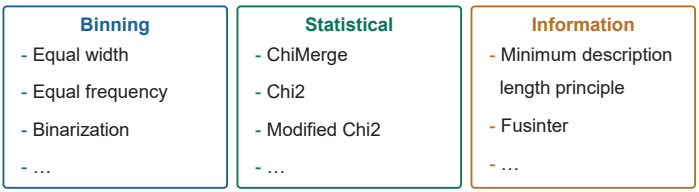


Figure 2-22: Simplified excerpt of data discretization techniques according to [GARC15, pp. 254-258]

In case of binning, the evaluation measure degenerates into basic constraints. Binning based on *equal width* is performed when all intervals have the same length. Another approach is *equal frequency*, which exhibit the same number of data points for every interval. [GUPA19] A statistical evaluation measure is, for instance *chi2*, while information-based examples are *minimum description length principle*, or *fusinter* [GARC15, p. 253]. However, both methods from statistical and information based data discretization are not commonly applied in production through the lack of transparency. A long list of 17 DPP methods for data discretization can be taken from Annex A.2.

Data reduction

Although new sensor systems increase the number of features and possibly provide more information for production use cases [ZHEN18b, p. 137], the amount of data required within modelling increases exponentially when introducing further sensors that eventually leads to the curse of dimensionality [BELL61]. High amount of training data is required to ensure that there are several instances with each value combination. When adding more features, data becomes sparse. The curse of dimensionality fosters overfitting [YIUT19]. Consequently, an increase in dimensionality by adding new features degrades ML models' performance after a certain point. This effect is called Hughes phenomenon or peaking phenomenon [HUGH68].

Data reduction performs a reduction on the original data set size [GARC15, p. 13]. This is achieved by removing irrelevant attributes that decreases data set's complexity and enhances the training efficiency during ML modelling [SAEY07, p. 2507]. Especially for large data sets, which may occur in production [PHAM05, p. 405], dimensionality reduction [POPO17] or feature selection [MOLD17] may be useful. On the contrary, reducing features also results in loss of information. In the following, data reduction will be divided into feature reduction through dimensionality reduction and feature selection and instance reduction through instance selection (see Figure 2-23).

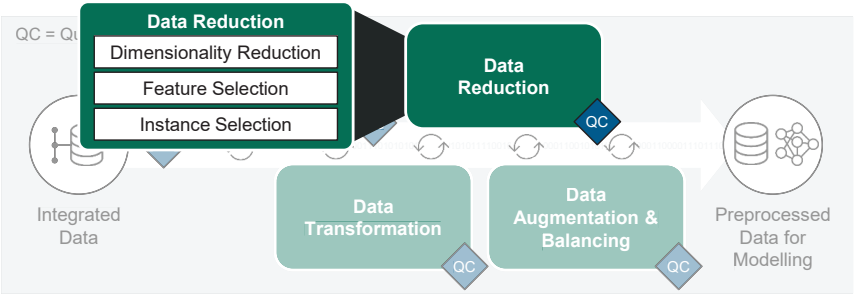


Figure 2-23: DPP categories of data reduction

Data reduction – dimensionality reduction

Dimensionality reduction is the process of reducing the number of attributes by generating a new set of features of an existing larger set of variables, which can be achieved components-, projection-, or ML-based as illustrated in Figure 2-24. Components-based dimensionality reduction aims at creating components or discriminants on a lower dimensional data space while retaining most of information in the data set. Prominent examples are *principal component analysis* (*PCA*) or *linear discriminant analysis* (LDA) [JOLL02], [COHE14]. Projection-based dimensionality reduction such as *locally linear embedding* (*LLE*) or *t-distributed stochastic neighbor* (*t-SNE*) are more complex

methods and commonly used for visualization purposes in case of high dimensional data [ROWE00], [MAAT08]. Although already developed in the 1980's, *autoencoders* represent newer approaches for dimensionality reduction [RUME86]. A long list of 19 DPP methods for dimensionality reduction including comprehensive descriptions, can be taken from Annex A.3.

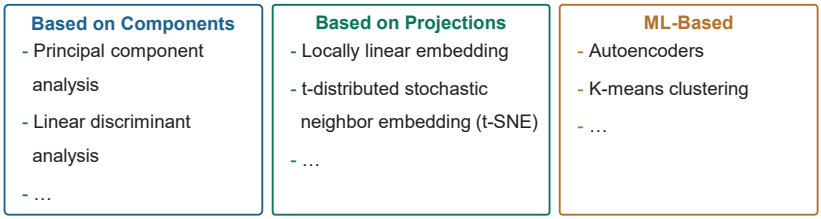


Figure 2-24: Classification of dimensionality reduction techniques

Data reduction – feature selection

Unlike dimensionality reduction, feature selection aims at selecting most relevant features in the data set at hand [MUSH19]. Feature selection can be divided into filter, wrapper, and embedded approaches. Figure 2-25 shows an excerpt of feature selection techniques. Exemplary filter approaches range from *missing value ratio* over *low variance filter* to *high correlation filter* [FANC21, p. 5]. The *low variance filter* requires feature scaling, however, a *StandardScaler* scales all features to a uniform variance, which is close to 1, resulting in a filter occasionally not identifying variances under a threshold of 0.6 [WIDM15] [BHAN20]. Based on given thresholds such as the correlation between or variance within attributes, features are removed. While filter approaches solely concentrate on the data set, wrapper methods include baseline ML algorithms to determine feature's impact on ML model performance. Features that do not increase the performance of the ML model are disregarded. [GARC15, pp. 174-175] Both *forward feature selection* (*FFS*) and *backward feature elimination* (*BFE*) as well as *recursive feature elimination* (*RFE*) represent prominent examples for wrapper methods [FANC21, pp. 5-6]. Besides filter and wrapper methods, feature selection can be embedded within the ML algorithm. *Decision trees* apply pruning to reduce the size. One or more splits are removed if the sections do not contribute to the model performance. Besides for *decision trees,* regularization is realized in *LASSO regression* by applying penalization terms to minimize coefficients of the equation towards zero leading to the removal of the corresponding attribute. Annex A.3 provides a long list of 26 DPP methods for feature selection including comprehensive descriptions, its advantages and disadvantages.

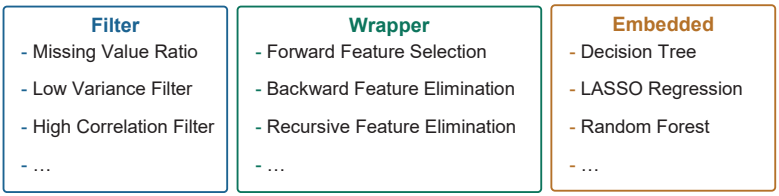


Figure 2-25: Classification of feature selection and excerpt of techniques

Data reduction – instance selection

Another common challenge is to select stratified and representative samples of the overall data population. To reduce the size of the data set in terms of instances and select appropriate data, instance or numerosity selection is performed. Efficient sampling is a critical requirement since it increases data quality and ML models trained on this sampled set can be scaled up easily. [KOTS06] Instance selection can also be divided into filter and wrapper methods. *Manual sampling*, also quantitative sampling, performs the instance selection manually by a production expert [FANC21, pp. 5-6]. Given the example as stated in Chapter 2.1, Blisks are milled in different operations and block-wise. The operation can be distinguished in roughing, pre-finishing, and finishing. Every blade is divided in several blocks to reduce vibrations during milling. Instead of using the whole data set, representative data sets can be taken from one block of a specific blade during one specific operation. This approach is easy, powerful and transparent, however, can also be very time consuming. Further filter approaches like *nearest sub-class classifiers*, or wrapper methods, such as *condensed nearest neighbor* are neither popular nor commonly used in practice. An excerpt of DPP methods can be found in Figure 2-26. A list of 16 DPP methods for instance selection can be taken from Annex A.3.

Figure 2-26: Classification of instance selection and excerpt of techniques

Data augmentation & balancing

On the one hand, more and more customized products results in lower number of data sets available for similar products [PILL22]. These phenomenon exacerbate the application of data driven projects, e. g., product quality prediction or anomaly detection. On the other hand, introducing new sensors lead to the curse of dimensionality. Then,

too few instances are available for training the ML algorithm successfully, while ML algorithms may require lots of data for generalization. Thus, one of the most reliable ways to improve performance is to get more data. [GAND21], [LYAS20] Data augmentation aims at artificially adding new instances to the data set at hand by means of DPP methods instead of further recording real production data [FANC21, pp. 10-11]. Besides augmenting data, rows are further managed by different balancing techniques. The following paragraphs focus on augmenting the samples and adjusting the samples to balance the target variable (see Figure 2-27).

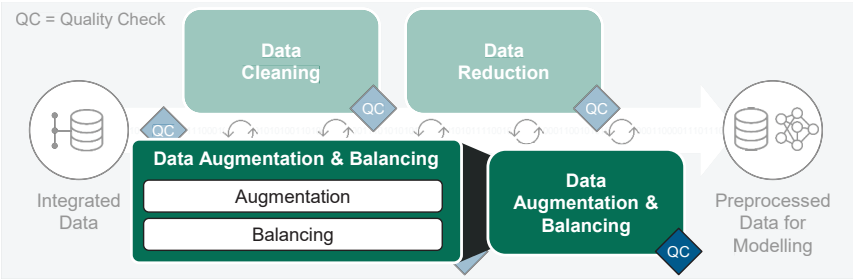


Figure 2-27: DPP categories of augmentation and balancing

Data augmentation & balancing – data augmentation

For numerical data, data can be augmented by applying *random noise* or *autoencoders* [BISH06, p. 347]. For this purpose, new data points are artificially added based on already existing data points. Adding a minor proportion of noise can make the ML model more robust to overfitting [BISH06, p. 347]. The noise is generated by adding predefined mean and standard deviation values to the original data set. Whereas too little noise has nearly no effect on the performance of ML models, too much noise or a tool large range of noise makes it hard to learn the underlying function, thus leading to overfitting [GARC15, p. 22]. Besides *random noise*, *autoencoders* can be used to augment the data [FANC21, p. 12].

Data augmentation & balancing – data balancing

Within production, especially in product quality prediction tasks as shown in Chapter 2.1, the data is highly imbalanced, i. e., the representation of classes is highly different. In case a certain class in the target column is underrepresented, ML models tend to fail when classifying the label [BATI04, p. 21]. Exemplarily, a production manufactures 100 products of the same type. At the end, the quality assurance decides, whether the product quality is within or out of specification representing a two-class classification. If a production manufactures 95 of 100 parts correctly, the resulting class distribution is 95:5 or 19:1. ML algorithms generalize poorly on highly imbalanced data,

which requires data balancing. Thus, balancing can increase data quality in a production context [MOLD17] by adjusting the amount of data, which on the contrary may introduce noise or resulting in the effect of curse of dimensionality. [BATI04, p. 21] Balancing can be realized through over-, under-, and hybrid sampling, whose corresponding methods are introduced in Figure 2-28.

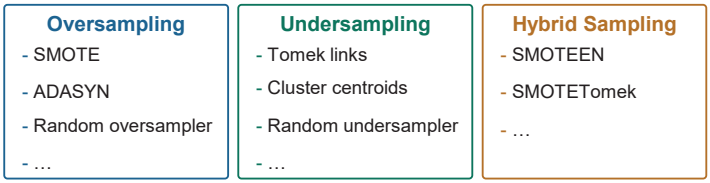


Figure 2-28: Classification of data balancing and excerpt of balancing techniques

Oversampling techniques such as *synthetic minority over-sampling* (*SMOTE*) or *adaptive synthetic sampling* (ADASYN) thereby increases the number of samples of the minority class. For the case of binary classification, the number of samples of the majority class is reduced during undersampling, which has the disadvantage that relevant information for the ML algorithm is lost. Common undersampling techniques are *random undersampler* or *tomek links*. When oversampling and undersampling strategies are combined, hybrid sampling is applied. Exemplarily, *SMOTEENN* oversamples the data first with *SMOTE* and subsequently undersamples the interim result by applying *edited nearest neighbor* (ENN). A list of 23 DPP methods for data balancing including comprehensive descriptions can be taken from Annex A.4.

Nowadays, balancing methods are also tailored for regression tasks that considers that the data distribution of a continuous values is skewed, while regions of interest, e. g., outer regions in the distribution, may be underrepresented. [KUNZ20], [BRAN17, p. 37]. One such balancing method is *Synthetic minority over-sampling technique for regression with gaussian noise* (*SMOGN*) [BRAN17, p. 37]. As for hybrid sampling in classification tasks, a combination of over- and undersampling is realized [BRAN17, p. 39]. The concept of *SMOTE* is applied for oversampling [KUNZ20], while *box plots* are used for outlier detection to find the data points being underrepresented [KUNZ19]. A subsequent undersampling is performed by disregarding those data points that are close to the mean of the entire data distribution [KUNZ20]. Unlike balancing techniques for classification, *SMOGN* additionally applies *Gaussian noise* with a mean of 0 and a standard deviation of 1 to obtain a more uniform distribution [BRAN17, p. 39], [BROW18]. In practice, *SMOGN* fails if no outliers are available in the target feature. Another example of balancing regression target is *SmoteRegress*.

Interim conclusion

In summary, the presented flowchart illustrated in Figure 2-14 serves as a structure, which is understandable, intuitive, and transparent. The DPP methods can be assigned to the different DPP steps of cleaning, transformation, reduction as well as augmentation and balancing. Each individual step is divided in categories such as missing value, outlier and noisy data handling within cleaning. Within each category, further structure has been provided to eventually assign DPP methods. The overall structure recommends a data quality check to determine the success of the DPP method at hand, while being highly iterative. In practice, DPP methods are not solely used, but combined to DPP pipelines. The unprocessed data set is handled by the first method in a pipeline and each subsequent method in the pipeline is applied asynchronously to the data set processed by the previous method [GÉRO18, p. 36] . The next subchapter therefore shows the approach and complexity of the creation of suitable DPP pipelines in production.

2.4.4 Creation of suitable DPP pipelines

Since it is a manifold task, creating suitable DPP pipelines forms one of the fundamental challenges data scientists face, when it comes to developing ML projects. [GIOV21, p. 1], [REDM18]. A configuration of DPP methods ends up in thousands of possible DPP pipelines. The following reasons make the creation of DPP pipelines especially challenging:

- *Use case specificity:* Dependent on the use case and its given data set, many versatile considerations need to be made. As stated in Chapter 2.4.2, requirements are placed on DPP through data set characteristics. The data set highly influences the choice of DPP methods. For instance, in case missing values are in the data set and can be classified as MAR, multivariate imputation methods should be applied. This only narrows down the number of possible DPP methods. However, the choice of one of these multivariate imputation techniques shown in Annex A.1 needs further be made due to the *no free lunch theorem* [WOLP97].
- *Complex search space:* As found in Chapter 2.4.3 and Annex A.1-A.4, there are hundreds of possible DPP methods to choose from. The search space becomes even larger, if not only single DPP methods are searched, but suitable DPP methods are combined to DPP pipelines [GÉRO18, p. 36]. Finding DPP pipelines that perform well for the problem at hand is tedious, since one needs to iterate over many possible solutions [ABDA17, p. 318].
- *Conditionality:* Certain DPP methods require the application of DPP methods beforehand (see Chapter 2.4.2) [PYLE99, p. 110]. As presented, *PCA* requires the centering of data beforehand through the application of standard scaler [FANC21,

p. 7]. Another example is the application of a power transformer prior to LDA [COHE14]. Furthermore, using specific DPP methods result in non-usability of other DPP methods. For example, dependent on the thresholds of the *low variance filter*, they do not work with previously scaled data.

- *Variance-Bias-Tradeoff:* DPP pipelines directly influence the variance bias tradeoff introduced in Chapter 2.3.2. Noise introduces variance in the data, whereas the removal of data through data reduction, undersampling, missing or outlier data handling results in loss of information. Data removal accelerates model training and can lead to better model generalization. However, if too much data is removed, the model will underfit. Therefore, noise can be inserted through data augmentation or oversampling, which can lead to overfitting through the insertion of too much variance. [GARC15, p. 22].
- *Iterative nature of DPP:* There are no fixed sequences of DPP pipelines. Methods that are in a sequence in the first place can be in a different order in the next use case. In consequence and due to the high use case dependency, DPP does not follow a structured procedure in practice. [ABDA17]
- *Stochastic behavior:* Some DPP methods exhibit stochastic behavior. If the scope is extended to DPP pipelines as in this thesis, the stochastic behavior is a major challenge. Stochastic behavior means that very similar inputs do not necessarily lead to similar results.
- *ML dependency:* Besides the dimension data quality characteristics, DPP is also influenced by the ML algorithms to be selected. As stated in Chapter 2.3.3, ML algorithms place requirements on DPP. They require specific data types and perform better with normally distributed as well as balanced data. [BROW20c]
- *Lack of a priori knowledge about DPP impact:* There are no default or standard DPP methods that should be chosen, and even less are sequences of multiple DPP methods generally recommended. The knowledge about whether DPP methods lead to the desired success can only be gained after training ML algorithms. Moreover, the quantification of the influence of single DPP methods is further complicated when combining DPP methods to pipelines. In practice, the phases of DPP and modelling are therefore run through multiple times. [GIOV21]
- *Limited experience*: The high specificity of use cases results in a shallow learning curve for data scientists. Since every new use case can be new in nature, only few aspects can be transferred to the new task. This phenomenon especially applies for production experts that attempt to build DPP pipelines.

As consequence, simple as manually exploring the search space by trial and error is tedious and tends to lead to unsatisfactory outcomes, since finding the suitable DPP pipelines requires knowledge about conditionalities, intuition about DPP's impact,

and a large amount of time. These reasons lead to the high proportions of 80 % of time spent on DPP in every ML project [CHAP00, p. 42], [GABE17]. The selection of DPP pipelines given production use case characteristics remains the major challenge in ML projects. Data scientists should be supported in selecting most performant, while efficient DPP pipelines. Decision support systems (DSS) have been developed in the last decades, which can be transferred to the problem in selecting suitable DPP pipelines for ML applications in production.

2.5 Decision Support Systems

The following sub chapter introduces forms and technologies of DSS. First, different forms of DSS, which have evolved in the past decades, are discussed in Chapter 2.5.1. Since developed DSS are empowered by different technologies, two most recent developed solutions are outlined, which are meta learning (Chapter 2.5.2) and automated machine learning (AutoML) (Chapter 2.5.3).

2.5.1 Forms of Decision Support Systems

Decision support systems (DSS) represent a computer based system participating and supporting in decision making processes of humans [POWE02], [SPRA82]. Although being already introduced in the mid-1950s, DSS yet lack a clear definition [KEEN80, pp. 2-3]. Depending on the problem and environment, demands and requirements for DSS differ. However, in general, it is an interactive computer information system that can help the decision maker to use data and models to solve problems [POWE02]. Over the years, different types of DSS have evolved, while architecture of DSS is designed on three main components: *data base*, *model*, and *user interface* [SPRA82], [POWE02]. In addition, *users* play a major role when designing a DSS [HAET99], [MARA99]. Since there is no clear definition, different taxonomies have been presented. DSS types can be distinguished by the target group, which can be personal, group, or organizational support, or the purpose that can either be data-driven, communications-driven or model-driven, among others.

According to ARNOTT & PERVAN, DSS types are personal, business intelligence, group, negotiation, intelligent, and knowledge management-based DSS [ARNO14, p. 275]. POWER distinguishes between communications, group, data, document, knowledge and model-driven, web-based and interorganizational DSS [POWE02, pp. 103-192] Each system attempts to meet individual demands, but in practice are often combined to cover all requirements for more complex problems [ARNO14]. In the following, the taxonomies that are mostly related to this work are outlined according to ARNOTT & PERVAN [ARNO14, p. 275].

- Personal DSS are small systems developed for a single user or a small number of independent users. A production example is represented by "DSS for production planning", in which production experts can create an optimal manufacturing plan [TSUB95].
- Group DSS facilitate the effective work of groups using communication-driven DSS for handling information. A group DSS is designed to maximize efficiency in decision making while minimizing the risk of information loss [JESS91].
- Data-driven or business intelligence-DSS are based on data and analytics to support decision making at all levels of an organization on large-scale. Business intelligence systems are often based on a data warehouse or data mart system [THOM15].
- Knowledge management-based or knowledge-driven DSS supports decision making by saving, retrieving, transferring, and applying the knowledge. Knowledge-driven DSS facilitate the access of cross-group knowledge.
- Intelligent decision support systems (IDSS) apply AI techniques to decision support. For instance, an IDSS was developed for the optimization of planning in production using ML techniques. The DSS integrates all production-related information, and creates a plan for implementation [GONZ20].

Due to recent progress in the field of AI, IDSS has attracted most attention. The basic idea is to apply AI techniques as *model* of the DSS architecture to develop a system, which supports decision makers by collecting and analyzing facts, detecting and diagnosing problems, as well as proposing possible actions. In addition, IDSS can be used to learn from experience. In the last decade, boundaries between main groups of DSS have disappeared, leading to further types of DSS combining different DSS types. [ZHOU08] When recommending DPP pipelines, a small number of users will be involved, therefore, a personal DSS will be designed. Moreover, due to high flexibility, and retrainability, IDSS are chosen leading to a combination of a personal and intelligent DSS.

For supporting in manual decisions, SERBAN ET AL. introduced intelligent discovery assistants (IDAs) aiming at assisting users in the data science process. IDAs represent a combination of IDSS and personal DSS and can be grouped in five categories differing in *data base, model* and *user interface* that are briefly described in the following. [SERB13]

- *Expert systems* support users through accessing knowledge from a knowledge base by executing an inference engine. The knowledge base contain previously acquired expert rules, while the inference engine aims at proposing suitable ML algorithms or DPP pipelines from the knowledge base. Given a new data set, knowledge is retrieved by the inference engine. [FEIG82] For instance,

SPRINGEX apply expert rules after a comprehensive questionnaire to the user. Then, the inference engine infers suitable classification algorithms. [RAES92, pp. 55-59]

- *Case based reasoning systems* memorize ML algorithms, or DPP pipelines respectively. Given a new use case, the case base can provide DPP pipelines based on similar use cases based on historical data. Various examples for case based reasoning systems exist [ENGE96], [LIND99], [MORI02].
- *Planning-based data analysis systems* design ML pipelines by formally defining input, output, preconditions, and effects of every method of the system, i. e., DPP methods and ML algorithm to be used. Operators are then combined to pipelines. The formal definition are thereby often ontology-driven [ZAKO11], [DIAM09].
- *Workflow composition environments* offer a graphical interface to manually build ML pipelines, in which the data flow is visualized. Examples range from Weka over RapidMiner to KNIME [HALL09], [MIER06], [BERT09]
- *Meta learning systems* use meta features of a given use case, e. g., data types of a tabular data set mentioned in Chapter 2.4.2, to propose a predefined target. Meta features serve as independent variables, while the target variable, i. e., DPP pipelines or ML algorithms, represent the dependent variable. The training process results in a supervised learning approach, mostly represented by a classification task. [VANS19] However, the choice of ML algorithms used in meta learning is also subject to the *no free lunch theorem* [WOLP97]. One example for meta learning systems for assisting in the data science process is the “data mining advisor” [GIRA05].
- Not mentioned by SERBAN ET AL., but recently used are *Automated machine learning (AutoML) systems*, which automatically train ML pipelines consisting of DPP pipelines, ML algorithms and optimized hyperparameters [HUTT19]. AutoML frameworks were developed to support the user, i. e., data scientists, in achieving performant ML pipelines within a reasonable amount of time.

Expert systems rely on expert rules, which are hard and tedious to acquire, while expert rules need to be updated in case of a change in requirements. Moreover, expert rules are subjective. A high number of expert rules need to be recorded, which can also lead to contradictory specifications. Case based reasoning systems involve experts when maintaining the case base. Thus, these systems are prone to manipulation and human error. Although being performant in extrapolating, planning based systems require formal definition of input, output, prediction and effects. Referring to hundreds of DPP methods and dozens of ML algorithms, the solution space becomes very high, and consequently, the formal definitions very tedious and non-manageable. Workflow

composition environments do not provide recommendations but only support in visualizing data flows. Therefore, workflow composition environments are not suitable for recommending DPP pipelines. Since meta learning systems and AutoML systems represent two promising solutions for the design of a DSS, both technologies are outlined in the following.

2.5.2 Meta Learning as DSS Technology

Meta learning, or “learning to learn”, comprises the concept of learning from experiences over many different tasks to recommend a most suitable output for a new task [VANS19, p. 35]. Due to the wide applicability, multiple types of meta learning have been developed in the field of ML. Ensemble learners use outputs from previous models as input to generate a less biased prediction. Hyperparameter settings of an ML algorithm are optimized using algorithms on a meta level. In multi-task learning, a meta model is trained on multiple similar tasks simultaneously. In this work, however, meta learning includes any type of learning, which is based on experience from prior tasks. Therefore, a meta target (I) is predicted based on a new use case. From every use case, meta information can be extracted as meta features (II), for instance, the number of outliers in a data set. Based on meta features, the meta target can be predicted using a meta model (III). Figure 2-29 shows the meta learning concept. In the following, each component of the meta learning concept is introduced.

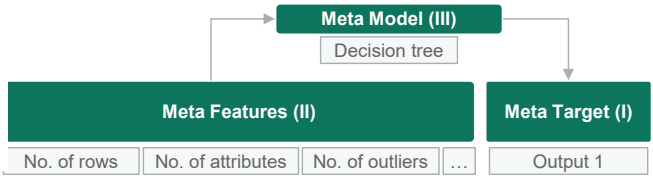


Figure 2-29: Meta learning concept with its three main components

Meta target (I)

The meta target represents the target variable to be predicted on. Dependent on the meta learning approach, the meta target can be a set of hyperparameter configurations, or an ML algorithm. In this work, the meta target is represented by a set of DPP pipelines.

Meta features (II)

Three approaches of acquiring meta data are defined by VANSCHOREN, which assumes that the meta target consists of an ML algorithm and its hyperparameter configuration [VANS19]. First, meta data can be acquired based on task properties to determine task similarities and relate ML model performances to input data. The second approach

describes the extraction of meta features by learning from historical model evaluations. Prior model evaluations, e. g., the accuracy of a classifier, are used to predict a suitable hyperparameter configuration for a new task. Lastly, VANSCHOREN discusses the technique of learning from prior ML models on how to transfer trained model parameters, which is not focused in this work. [VANS19] In the following, the approaches of learning from task properties and model evaluations are outlined.

Through the acquisition of different groups of meta features, task properties can be determined. General-purpose meta features provide information about the data set shape or number of outliers. BENSUSAN ET AL. then distinguishes between three kinds of task characterizations [BENS00a].

- *Statistical and information-theoretical* meta features can be extracted from the data set at hand. These meta features are calculated on single features, e. g., the variance of the feature, or combinations of features, e. g., the correlations between two features. When correlations are calculated, outputs are aggregated by descriptive statistics such as mean, or standard deviation. While statistical meta features are results of numerical attributes in the data set, information-theoretical meta features focus on categorical features. In this case, measures such as entropy, information gain, or signal to noise ratio are computed. [MICH94]
- Besides general-purpose meta features, *landmarking* is used to describe a task. Landmarking relies on the assumption that the error rates of a baseline model in its simplest form provides information about the task complexity. In addition, the error rates give insights, for which tasks a baseline model is performant, while indicating the performance of more complex models. Landmarkers used are decision, worst, and random nodes. Decision nodes, also called best nodes, represent the root of a *decision tree* aiming at maximizing the information gain when splitting based on an attribute, while worst nodes minimize the information gain to explore the search space as best as possible. The random node selects the attribute to be split on randomly. Moreover, 1-NN, elite-NN are commonly applied as well as Naïve Bayes and discriminant analysis. [BENS00b]
- *Model-based* meta features apply *decision trees* to extract information. Model parameters of a *decision tree* are then related to the data set. Model-based meta features assume that the complexity of a *decision tree* give insights about the complexity of the task. An exemplary measure is nodes per attribute or nodes per instance. [VILA04, p. 35], [PENG08]

Based on learning from task properties, historical model evaluations can serve as meaningful information for a meta model. In contrast to landmarking and model-based meta features, model evaluations consider historical performances of the meta target.

In combination with further meta features, model evaluation provides information about which configurations were performant in the past. Lastly, domain-specific knowledge is used in practice to generate further meta features. [VANS19]

Meta model (III)

The extracted meta features serve as input variables, while the meta target represents the target variable. Consequently, the ML training process introduced in Chapter 2.3.2 is performed. Since meta learning systems are used for recommending outputs in a ranked manner, a meta model needs to be selected that provide a ranked recommendation. Common algorithms being used are tree-based algorithms, ensembles, or adaptions of *K-NN*. [BRAZ09] Newer approaches also use automated machine learning (AutoML) systems to recommend ML pipelines, which is introduced in the following.

2.5.3 Automated Machine Learning as DSS Technology

The ML pipeline in production introduced in Chapter 2.2 consists of the phases data integration, preparation, modelling and deployment. Within the phases, many manual decisions need to be carried out by the data scientist. For instance, ML algorithms need to be chosen dependent on use case requirements causing a tedious, error-prone, and time consuming endeavor. For this reason, AutoML aims at supporting to find performing ML pipelines automatically. [HUTT19] The goal of AutoML platforms lies in optimizing classification, or regression scores by finding best performant pipelines, which consist of ML algorithms including its hyperparameters. This challenge is also known as *combined algorithm selection and hyperparameter optimization* (CASH) problem. Within a learning process, best configurations of ML algorithms and hyperparameters are searched. [FEUR15, pp. 115-117] Developed AutoML systems such as *auto sklearn* or *TPOT* are validated on 100 to 150 use cases [HUTT19, pp. 113, 151]. SHANG ET AL. use 220 classification and 80 regression use cases to validate the AutoML-based system called *Alpine Meadow* [SHAN19, p. 10]. Although providing promising results, AutoML systems do not consider the whole ML pipeline in production but are mainly focusing on modelling and partially data preparation phase.

Due to the various application possibilities, AutoML can be used in three ways. First, AutoML can be used as learning concept within a meta learning system. Secondly, AutoML itself can serve as a DSS for recommending DPP pipelines. Thirdly, the result of the recommendation can also serve as a warm start for AutoML systems to find best performing pipelines given the starting point of the recommendation and again facilitating the use of DPP and ML in production.

2.6 Interim Conclusion

In this chapter, the fundamentals in the recommendation of DPP pipelines for ML applications in production were introduced. In the beginning, ML application areas in production (Chapter 2.1) and process models for ML applications including the ML pipeline were outlined (Chapter 2.2). Chapter 2.3 consisted of ML approaches, learning tasks and ML algorithms used in this work. A detailed description of DPP (Chapter 2.4) was followed by the presentation of decision support systems (Chapter 2.5).

Within every chapter, the thesis focus was set, which is summarized in Figure 2-30. Since the focus lies on recommending DPP pipelines for ML applications in production, three application areas from production are covered. For the ML pipeline, DPP was introduced and classified in the steps of cleaning, transformation, reduction, as well as augmentation and balancing.

Criteria & Chapter		Characteristics				
Application areas	2.1	**Product**	**Process**	**Machines & assets**		
ML pipeline	2.2	Integration	**Preparation**	Modelling	Deployment	
Data preparation	2.2	**Preprocessing**	Feature engineering			
Data preprocessing	2.4	**Cleaning**	**Transformation**	**Reduction**	**Augment & Balancing**	
Learning approach	2.3	**Supervised learning**	Unsupervised learning	Reinforcement learning		
Learning task	2.3	**Classification**	**Regression**			
Q-characteristics	2.4	**Inherent**	Inherent-system-dependent	System-dependent		
Q-management	2.4	Process-oriented	**Data-oriented**			
Data type	2.4	**Tabular**	Image	Audio	Text	Video
Inner data structure	2.4	**Time series data**	**Cross sectional data**	**Panel data**		
Data scales	2.4	**Ordinal**	**Nominal**	**Ratio**	**Interval**	
DSS	2.5	Group	**Personal**	Knowledge	Document	**Intelligent**	…
Technology	2.5	Expert system	Case-based	Planning-based	Workflow	**Meta learning**	**AutoML**

Q = Quality DSS = Decision Support System

Figure 2-30: Morphological box to visualize the thesis focus

Since DPP is highly dependent on the ML algorithm at hand, a focus is put on supervised learning while considering both regression and classification tasks. In terms of data quality, inherent quality characteristics are taken into account. DPP thereby manages quality of the data in a data-oriented manner. Tabular data will be considered in

the dimensions of time series, cross sectional, and panel data as well as ordinal, nominal, ratio, and interval in scale. Lastly, a combination of intelligent and personal DSS combined with AutoML or meta learning represent promising approaches. Given the fundamentals and thesis focus, requirements can be derived (Chapter 3.1) for the recommendation of DPP pipelines for ML applications in production. Based on Chapter 3.1, existing approaches (Chapter 3.2) can be analyzed to formulate the need for further research and to underline the goal of this thesis.

3 Derivation of Requirements and Analysis of Existing Approaches

Based on the fundamentals, requirements can be defined, which need to be met by the DSS to be developed. Chapter 3.1 therefore comprises the derivation of requirements for the DSS, which aims at proposing suitable DPP pipelines for ML applications in production. Chapter 3.2 then describes existing approaches for recommending DPP pipelines. Finally, it is assessed to what degree the existing approaches fulfill the derived requirements. Based on these results, the research gap is outlined and necessary actions are derived.

3.1 Derivation of Requirements

The overall goal is to develop a DSS capable of recommending suitable DPP pipelines for supervised ML applications and tabular data from production use cases. Within the thesis focus described in Figure 2-30 and challenges outlined in Chapter 2, requirements for the DSS can be deduced. The requirements, which are displayed in Figure 3-1, are classified in fundamental prerequisites (Chapter 3.1.1), functional requirements (Chapter 3.1.2) and user-specific requirements (Chapter 3.1.3).

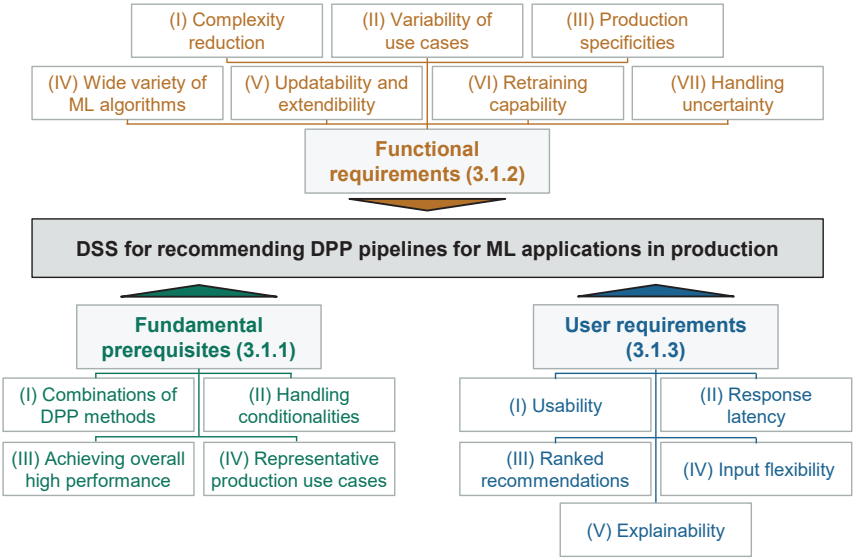


Figure 3-1: Overview of requirements placed on the DSS

3.1.1 Fundamental Prerequisites of DSS

For creating a DSS that recommends DPP pipelines, fundamental prerequisites must be met. Instead of recommending single DPP methods, the focus needs to be put on recommending fully comprehensive pipelines. To be capable of recommending DPP pipelines, best performing DPP pipelines need to be available. Best performing pipelines need to be found given representative production use cases in terms various data set characteristics and ML algorithms.

(I) Recommending DPP pipelines considering combinations of DPP methods

To enable ML for production use cases, combinations of DPP methods need to be applied (Chapter 2.4.4). Therefore, the DSS needs to offer DPP pipelines with a variable number of DPP methods, which contain methods from every DPP step and category presented in Chapter 2.4.3. The DSS must be capable of recommending DPP pipelines, while one DPP pipeline can consist of only one DPP method or multiple methods depending on use case characteristics. If the DSS is able to provide DPP pipelines consisting of a variable number of DPP methods considering DPP steps and categories, the requirement is fulfilled.

(II) Handling conditionalities

The conditionalities between DPP methods need to be considered when configuring DPP methods into DPP pipelines. Moreover, the combination of DPP methods should be meaningful. Meaningfulness means that DPP methods are combined so that each individual DPP method has an impact on ML model performance as presented in Chapter 2.4.4. For instance, the combination of a *MinMaxScaler* and *VarianceThreshold* does not represent a conditionality issue. However, dependent on the variance threshold value, the *low variance filter* will not have any purpose. Thus, the DSS must consider these dependencies between methods. The requirement is fulfilled if the DSS considers DPP conditionalities and dependencies.

(III) Achieving high overall performance

Chapter 2.4.1 has revealed that high ML model performances are especially achieved only if suitable DPP methods are selected. For the development and prior to the application, DPP pipelines need to be identified for given production use cases that achieve higher ML model performance compared to performance when no DPP is performed. The best performing pipelines then serve as a basis for the DSS. In the application, the DSS needs to find well performing pipelines from the set of previously determined best performing pipelines. The requirement is fulfilled if DPP pipelines achieve higher

ML model performances on average compared to the performance if no DPP is applied in both the development and application phase.

(IV) Representative production use cases for setting-up the DSS

Due to the high complexity, profound recommendations of DPP pipelines can only be made if the data basis is large enough. For learning systems, at least 300 use cases for both regression and classification are deemed sufficient to set up a DSS according to the findings in Chapter 2.5. In addition, representative production use cases with versatile data set characteristics from production use cases and ML algorithms need to be considered (see Chapter 2.1, 2.3.3). The requirement is fulfilled, if at least 300 production use cases with different data set characteristics for both classification and regression are used to set up the DSS.

3.1.2 Inherent Functionality of the DSS

Since DPP is a complex and tedious task, the DSS needs to be designed to facilitate DPP. The DSS needs thereby be capable of handling versatile production use cases and general production specificities. Once in application, components should be updatable, extendable, as well as retrainable.

(I) Complexity reduction

The number of inputs being required to recommend DPP pipelines to the user must be as small as possible. Moreover, the number of DPP pipelines being recommended need to be in a manageable range to not overwhelm the user. The number of DPP methods within DPP pipelines should be minimized to reduce both complexity and computing time without significantly reducing the pipeline performance (Chapter 2.4.4). If two DPP pipelines achieve the same performance, the DPP pipeline with less DPP methods should be recommended. A reduced complexity fosters better understandability of recommendations by the user. The requirement is fulfilled if less complex DPP pipelines are selected if they achieve the same scores. In addition, it is fulfilled if the number of input features is kept as low as possible.

(II) Handling of use cases' variability

The DSS needs to be able to handle a wide variety of production use cases. Chapter 2.1 reveals that ML applications in production are characterized by versatile determining factors such as data set, and ML algorithms being used. This results in diverse use case characteristics. According to Chapter 2.4.4, different DPP pipelines consisting of heterogeneous DPP methods achieve overall best performances. Heterogeneous DPP methods are different in terms of the degree of data set's manipulation that lead to different ML model performances. To ensure high overall performance of the

DSS, heterogeneous methods need to be used to configure DPP pipelines. This requirement is fulfilled if the inherent functionality of the DSS relies on the combination of heterogeneous DPP methods.

(III) Coping with production specific requirements

As underlined in Chapter 2.1 and Chapter 2.3.2, many production use cases for ML are either classification or regression learning tasks. Thus, the DSS needs to handle both classification and regression as in-built functions. In addition, metrics used for assessing the performance of ML models should be tailored to production purposes. The requirement is deemed as fulfilled if the DSS comprises DPP pipelines for classification and regression and uses the performance metrics F1 score and RMSE to evaluate the performance of DPP pipelines.

(IV) Handling a wide variety of ML algorithms

To ensure high performance over a wide range of production use cases, a variety of ML algorithms should be considered, since DPP and ML highly depend on each other as discussed in Chapter 2.4.4. The choice of ML algorithms has a crucial impact on the choice of DPP pipelines due to their different robustness against data quality characteristics (see Chapter 2.3.3). As further discussed in Chapter 2.3.3, 18 supervised learning algorithms are most often applied in production. The requirement is fulfilled if 18 ML algorithms can be chosen by the user to achieve the highest performance for a given production use case.

(V) Updatability and extendability

It is required that the DSS can be updated and extended. Updatability means that obsolete DPP methods can be removed, or new versions of DPP methods incorporated into the DSS. As discussed in Chapter 2.4.3, new DPP methods and ML algorithms are introduced in a frequent manner, which requires an updatability of the DSS. In addition, the user should have the opportunity to add further DPP methods to the DSS but also remove obsolete DPP methods. The same applies for ML algorithms being used. The requirement is fulfilled if the user can update and extend the DSS by adding or removing DPP methods and ML algorithms.

(VI) Retraining capability

The DSS should be able to learn from new data sets and use cases, which are provided by users, and which can also be inserted for further development. As presented in Chapter 2.5, the DSS can be developed on the basis of learning systems such as meta learning. For this reason, the system should be retrainable to adapt to new use cases and increases its performance. The requirement is fulfilled, if the DSS is retrainable.

(VII) Handling uncertainty

Uncertainty stems from various sources. First, the user who applies the DSS inserts uncertainty by providing imprecise inputs. Second, the developer brings in uncertainty during the development of a DSS, when following a data-driven approach. The stochastic behavior of DPP methods and ML algorithms results in further uncertainty (see Chapter 2.4.4). For that reason, uncertainty needs to be considered and handled by the DSS. The requirement is fulfilled, if the DSS considers uncertainty during development and provides information about the uncertainty of a recommendation.

3.1.3 User-Specific Requirements

For the involved experts and disciplines in production, the usability represents one key requirement of the DSS. Since results are preferred to be available on short notice, DPP pipelines need to be recommended quickly by the system. Recommendations must have a ranked order to be capable of providing the user the opportunity for choosing between different DPP pipelines. Besides input flexibility, the possibility of selecting only explainable pipelines should be given to the user.

(I) Usability

The target group of the DSS are data scientists, production engineers, IT experts, and software engineers working with ML that need to preprocess data. For this reason, the system needs to be designed so that it is easy to use also for non-programmers, while considering the criteria of ML use cases in production. The DSS needs to be operable via a user interface. To further increase the usability, the outcome of the DSS should also serve as warm start for AutoML systems as described in Chapter 2.5. The requirement is fulfilled if the target group is enabled in interacting with the DSS.

(II) Response latency

To support the user in preprocessing data, the DSS should display recommended DPP pipelines as quickly as possible. A low response latency increases the user satisfaction. The requirement is fulfilled if the response latency is lower than five minutes. Five minutes is considered a reasonable compromise to allow comprehensive operations and present results in a short time.

(III) Ranked recommendations

Since there is not only one possible DPP pipeline, which could be used for a given use case, a ranking should be available to give the user the opportunity to try out different DPP pipelines. Therefore, the DSS should rely on a system that enables ranking. Thus, a ranked recommendation of DPP pipelines should be realized that consider only best

performing DPP pipelines. Moreover, ranked recommendations should also give information about relevant DPP methods. The requirement is fulfilled if the DSS provides a ranked recommendation.

(IV) Input flexibility

The DSS need to provide input forms for the data set, selectable ML algorithms and external requirements such as explainability to receive necessary information for the pipeline recommendation. Even if the user does not input every detail of the use case, the DSS should be able to recommend DPP pipelines. Missing details can stem from missing information about ML algorithms or explainability preferences. The requirement is fulfilled if the DSS provides a recommendation, although inputs about explainability or ML algorithms are missing.

(V) Explainability

Explainability is highly relevant in the production domain, especially for applications, which require high safety standards as discussed in Chapter 2.1. Thereby, explainability can have different dimensions. DPP methods or ML algorithms being used can be explainable. The requirement is fulfilled if the DSS draws on both explainable and unexplainable DPP methods and ML algorithms.

3.2 Analysis of Existing Approaches

Based on derived requirements in the previous chapter, existing approaches for recommending DPP pipelines for ML applications in production are analyzed. Until now, DPP is carried out mostly via trial and error [KRIS17, p. 12]. To facilitate the application of DPP, various approaches have been introduced. The approaches can be distinguished in three different areas dependent on the complexity and scope. First, basic approaches provide an overview over existing DPP methods without giving any recommendation. Therefore, Chapter 3.2.1 discusses basic approaches of cheat sheets and flow charts. The second area of approaches focuses on assisting the user in the choice of DPP methods and pipelines. These user assistance systems are built on different technologies such as meta or reinforcement learning that are introduced in Chapter 3.2.2. Ultimately, as approaches of AutoML comprise DPP methods and aim at performing DPP automatically when applying ML algorithms, AutoML systems are investigated in more detail (Chapter 3.2.3).

3.2.1 Basic Support in Preprocessing Data

Nowadays, DPP is mainly performed in an iterative manner through intuitive decision making and trial and error by data scientists [KRIS17, p. 12]. To structure DPP and to

provide an initial support, cheat sheets and taxonomies in form of flow charts have been developed. Cheat sheets focusing on DPP give advice on how to implement different DPP techniques dependent on the programming language and library. The high number of cheat sheets only present DPP methods in excerpts, thus, are often incomplete for sake of clarity. Examples of cheat sheets are pandas data wrangling, or scikit-learn cheat sheets [DATA22b], [DATA22a]. Even though cheat sheets provide a first assistance in preprocessing data, they do not focus on recommending DPP pipelines dependent on a production use case.

Another approach for supporting data scientists in DPP is to depict the preprocessing in a flow chart. Based on a sequence of questioning and answering, DPP techniques are recommended. For instance, PANDEY & PRADHAN suggest a flow chart consisting of cleaning, integration, transformation, reduction and discretization. Questions are raised whether the data needs to be cleaned and if confirmed, a set of techniques are proposed. [PAND14]

In addition, the CRISP-DM that has been introduced in Chapter 2.2 presents five subphases within the data preparation step, which are “select”, “clean”, “construct”, “integrate”, and “format” data. Besides the description of tasks and activities, no further focus is put on techniques to be selected. [CHAP00, pp. 42-47] Flow charts in general lack the recommendation of pipelines dependent on specific use cases and only serve as a first structuring of DPP. For that reason, user assistance systems have been developed that attempt to aid data scientists in performing DPP.

3.2.2 User Assistance Systems for DPP

Potter’s Wheel by RAMAN ET AL. (2001)

Potter’s Wheel, which has been introduced in 2001, comprises an interactive data cleaning tool assisting the user in applying basic data cleaning methods. Exemplary methods are *adding*, *dropping*, *merging*, or *dividing features* of data sets. Through a spreadsheet-like display, transformations can interactively be generated, while users can directly add or undo changes. Transformations are based on a discrepancy detector that determines any non-consistency in the data. When applying *Potter’s Wheel*, sequences of transformations are built in an interactive manner that end up into DPP pipelines, while users can specify desired results through the display. *Potter’s Wheel* can be extended by adding further methods and algorithms for discrepancy detection. [RAMA01]

Since the focus in *Potter’s Wheel* lies not primarily on ML, methods are limited to very basic data cleaning operations, which leaves out further mandatory DPP steps in case of ML applications such as transformation, reduction or augmentation and balancing.

Since users need to specify transformations on their own, experience is required to perform target-oriented transformations that take a lot of time. *Potter's Wheel* also does not introduce a learning approach to optimize its performance based on new input data, thus, retraining capability is not provided. Uncertainty is left out of consideration.

ActiveClean by Krishnan et al. (2016)

A stronger focus on data cleaning for ML applications is put within *ActiveClean*. As an extension of SampleClean [KRIS15], *ActiveClean* aims at modelling the data cleaning problem using *SGD*, whose concept has been introduced in Chapter 2.3.2. After initialization, the *sampler* iteratively identifies batches of faulty data. A *detector* can further be specified that consists of either a rule-based or an adaptive detector to be capable of identifying dirty data successfully. Once determined, the *cleaner* cleans faulty instances by applying two methods, which are either *outlier removal* or *feature transformation*. Feature transformation in this context comprises the cleaning of samples that contain noise such as typos or NaN values. The cleaned data set is afterwards used to train an ML algorithm. An *updater* updates a model using the cleaned batch in a *SGD*-manner, which supports in serving convergence. This process is repeated until a stopping criterion, e. g., error convergence, is reached. An *estimator* further uses the updates from the *updater* to recommend and prioritize dirty instances to be chosen by the *detector*. [KRIS16]

Ultimately, *ActiveClean* provides a user assistance for cleaning data and subsequently train ML algorithms. *ActiveClean* combines different methods and iteratively improves the quality of the data set. However, only two methods have been outlined within the *feature transformation*, while other steps such as transformation are left out entirely. DPP methods need to be implemented solely by the data scientist. Only five data sets have been selected to identify the performance of the system not representing a solid data basis.

BoostClean by Krishnan et al. (2017)

Inspired by *ActiveClean*, *BoostClean* aims at achieving highest data quality using *adaptive boosting*, or *AdaBoost*, to recommend DPP methods (see Chapter 2.4.3). *BoostClean* consists of three key features. The first feature comprises a detector and repair library that provide necessary methods for detecting and overcoming data quality issues. To detect data quality issues such as outliers, the detector library is applied that contains methods such as *isolation forests* or *minimum covariance determinant* (see Annex A.1). The repair library provides DPP techniques such as *mean*, *median* or *mode imputation*. *BoostClean* then applies the detector library to identify a sub set of conditional repair methods. A DPP pipeline is constructed by iteratively adding further repair methods to the sequence. In between, a *random forest classifier* is trained

to identify the performance of the sequence of DPP methods. By applying *AdaBoost*, the scores are used to recommend the next repair method to be implemented. Main differences to *ActiveClean* are the optimization technique, which is *AdaBoost*, and the consideration of further DPP methods from data cleaning. The approach was validated on twelve open source classification data sets [KRIS17]

While only focusing on classification tasks and therefore not coping with production specific requirements that require regression, DPP methods of transformation, reduction and augmentation & balancing are not considered in *BoostClean*. In the course of this, *BoostClean* is based on twelve open source data sets, which represents only a small data basis to make a profound decision on the performance of *BoostClean*.

HoloClean by Rekatsinas et al. (2017)

HoloClean aims at cleaning data by considering integrity constraints, and external data sources. Integrity constraints are meant to ensure the consistency within the data when repairing inconsistent data. According to *HoloClean*'s nomenclature, the input comprises the data set, external information, denial constraints and matching dependencies. A production example would be a data set from the shop floor, in which a machine operator is assigned to the machine. A denial constraint represents a conditionality, so that a machine operator is always working only on one machine, while the machine is operated by three different operators in three shifts. Matching dependencies can be external information such as tools being used in the machine. This input is cleaned by *HoloClean* based on an error detection, compilation, and repair module. Within detection, *outliers* are detected, *integrity constraints* checked, and possible repairs identified. In compilation, candidates are compiled to a probabilistic program, while in the repair module, the probabilistic model is executed to eventually clean the data set. Through the probabilistic approach, uncertainty is handled. [REKA17]

In contrast to *BoostClean* and *ActiveClean*, *HoloClean* covers more input features to clean the data and is based on probabilistic inference instead of *SGD* or *AdaBoost* to find best performing cleaning techniques. Therefore, *HoloClean* handles conditionalities and various use cases better. However, *HoloClean* also only focuses on cleaning operations, while other DPP techniques from different steps are not covered. In addition, probabilistic inference brings in more complexity while being less transparent.

Further approaches have been introduced that assists in cleaning data for ML applications. Mahdavi et al. propose the so called *cleaning workflow orchestrator* that generates data cleaning workflows by first extracting meta features from a data set, detecting errors, and lastly generating data set-specific cleaning workflows. [MAHD19] Other approaches, which follow a similar approach, are *AutoSuggest*, or *Auto Data Cleaning*

[YAN20], [HAID20]. The *cleaning workflow orchestrator* is built for regression and classification tasks, represents an understandable inherent functionality, and thereby reduces complexity. However, it currently lacks representative production use cases that proves the performance, and does not cover both many ML algorithms and DPP methods from different DPP steps and categories.

Learn2Clean by Berti-Equille (2019)

In *Learn2Clean*, users are assisted in both preprocessing data and applying ML algorithms. *Learn2Clean* is based on reinforcement learning, in which Q-learning is applied to identify most performant DPP pipelines. A user selects an ML algorithm from classification, regression, or clustering based on which DPP methods are implemented in a sequence by the system. Once prepared, the previously chosen ML algorithm is trained. Given the score of the ML algorithm, accuracy of classification model, the actions, i. e., DPP methods to be selected, are optimized using Q-learning. Various DPP steps are included within *Learn2Clean*. In case of missing value handling, *K-NN*, *MICE*, *expectation maximization* and *most frequent value* imputation are proposed. Outlier handling consists of *interquartile range*, *local outlier factor* as well as *z score*-based outlier detection. Noise in the data is treated based on *exact and approximate duplicate handling*. Constraint discovery and pattern checking are used for determining consistency in the data. For scaling the data, *MinMaxScaler*, *z score* and *decimal scale normalization* are applied. Features are selected by applying *missing value ratio*, *removing correlated features* and applying a wrapper method with a tree-based classifier as baseline model. *Learn2Clean* was validated on three data sets. [BERT19] Based on *Learn2Clean*, *CleanEX* was introduced, which thoroughly investigates chosen DPP pipelines and ML algorithms regarding its transparency and which touches the handling of uncertainty [BERT21].

Learn2Clean does not focus on categories like augmentation and balancing methods, which are deemed highly important for final model performance as discussed in Chapter 2.4.3. Although being applicable to different learning tasks, provided scores for regression, MSE, and classification, accuracy, are not preferred in production context, since a combination is required between different performance metrics. The reinforcement learning approach is developed based on three data sets, whereas only one data set was used for regression purposes not representing a sufficient data basis. Moreover, Q-learning may be highly inefficient and computationally expensive, since the agent learns on a long term assessing each DPP pipeline including the ML algorithm in every iteration. Ultimately, *Learn2Clean* is highly complex and less transparent.

PRESISTANT by Bilalli et al. (2018)

In *PRESISTANT*, a meta learning system is developed to identify the impact of DPP methods on classification algorithms. Based on the input, a ranking of DPP methods is provided, which aims at having a positive impact on the ML model performance. By extracting meta features from the data set, determining ML model performance prior and after applying a sequence of DPP methods, the impact of DPP is determined. Thus, the relative performance obtained between a classification model before and after DPP is used as target variable. A meta model is then trained on the data set to identify the relationship between meta features of the data set and the relative performance increase of ML models based on implemented DPP methods. From there, *PRESISTANT* assists the user in reducing the number of DPP methods to be chosen by going through three phases. In the pruning phase, the number of DPP methods are reduced by applying expert rules. The second phase comprises the learning phase, in which a *random forest classifier* is applied as meta model that classifies, whether the pipeline at hand has a positive, negative, or no impact. In the third phase, results are ranked by the classes probability. According to its authors, *PRESISTANT* was evaluated on hundreds of data sets. [BILA18]

PRESISTANT does not focus on regression tasks, which are common in production when focusing on predictive maintenance or product quality prediction as seen in Chapter 2.1. Moreover, no focus is put on encoding, augmentation and balancing tasks, as they have an important impact on ML model performance and are specifically required in real world production use cases. The approach also leaves out the sequence of DPP methods, but DPP methods need to be iteratively identified, which ends up in a time consuming endeavor and reduced usability. Many meta features are to be extracted making the explainability of *PRESISTANT* more difficult and the system in total complex.

MetaPrep by Zagatti (2021)

Similar to *PRESISTANT*, Zagatti developed a meta learning system to find most performing DPP pipelines for classification tasks through the extraction of meta features. The meta learning algorithm is chosen as *1 nearest neighbor*. Eleven data preparation techniques are combined in a combinatorial approach, ending up in 180 possible pipelines, which are benchmarked on 80 data sets using a *random forest classifier*. The five best performing pipelines were selected into a pipeline pool that are further recommended by the system. Even though the selection of different ML algorithms has a fundamental impact on the choice of DPP pipelines, the approach lacks the opportunity of choosing different ML algorithms. [ZAGA21]

For finding most performant pipelines, accuracy is used as performance metric not leading to a profound performance estimate in real world production data sets. DPP techniques from imputation, encoding, feature scaling and balancing are applied, however, augmentation, power transformers as well as data reduction methods are missing. No ranking is applied to the best performing pipelines given a new data set. In addition, *MetaPrep* does not cover uncertainties.

RankML by Laadan et al. (2019)

RankML aims at recommending ML pipelines including various DPP methods through meta learning. Based on 244 benchmarked data sets, the meta learning system recommends the best performing ML pipeline for the specific learning task in a ranked manner. The pipelines, from which meta features are extracted, are generated using *tree-based pipeline optimization tool* (*TPOT*), which will be introduced in Chapter 3.2.3. Different types of meta features are extracted. First, data set related meta features such as descriptive and correlation-based information are considered. Secondly, meta features are used that provide information about the pipeline's topology of the pipeline. Given the meta features of the data sets and pipeline representation, the target variable, i. e., ML pipeline can be predicted using the meta learning algorithm *XGBoost*. In addition, *RankML* is retrainable. [LAAD19]

Even though several meta features are extracted, *RankML* misses newer approaches of meta feature extractors such as landmarkings. The system further lacks balancing as well as augmentation methods, which are highly relevant for production use cases. Since *RankML* is based on the pipelines generated by *TPOT*, it is less extendable and updatable in terms of incorporating new methods. The same applies for ML algorithms. *TPOT* already covers a set of ML algorithms but does not provide more than twelve ML algorithms.

As interim conclusion, many approaches can be found, which support the user in identifying DPP methods or DPP pipelines. Many of them combine the step of DPP with ML approaches. Different technologies are used to propose DPP methods or pipelines, which can be based on probabilistic inference, reinforcement, or meta learning. In this context, the concept of AutoML has gained popularity. AutoML systems increasingly cover DPP techniques. Therefore, the next sub chapter investigates, how comprehensively AutoML systems support DPP.

3.2.3 AutoML Systems and AutoML-Based DPP

As shown in Chapter 2.5, AutoML aims at democratizing the use of ML by supporting users through automatic generation of fully comprehensive ML pipelines. Low level decisions such as the choice of DPP methods and ML algorithms are undertaken by

the AutoML system. [FEUR22, p. 1] Many fully comprehensive AutoML frameworks have been introduced in recent years, which can be divided into proprietary, and open source tools. These frameworks attempt to find best performing ML pipelines focusing on the choice of ML algorithm selection including the best hyperparameter settings. Increasingly more DPP methods have been incorporated into AutoML systems in recent years. At the same time, systems have been developed that shift the focus from modelling to DPP and thereby transfer the large configuration space from ML algorithms to DPP methods and its hyperparameter settings. Consequently, fully comprehensive AutoML and AutoML-based DPP frameworks are introduced in the following based on their popularity and consideration of DPP.

Fully comprehensive AutoML: proprietary systems

Azure Machine Learning from Microsoft represents a proprietary service that facilitates AutoML for both classification and regression tasks. The platform offers DPP methods and ML algorithms, while optimization is realized by applying Bayesian optimization (BO). In terms of DPP, no methods could be found in the documentation. [MICR20] *Amazon SageMaker* from Amazon Web Services enables manual AutoML for regression and classification via the so called autopilot [AMAZ20]. The autopilot automatically implements cleaning, and transformation methods with ML algorithms. Optimization is realized through BO. Custom DPP pipelines can be generated through a python user interface. *Google AutoML Tables* provides a user interface for generating ML pipelines. However, no further information could be found on which DPP methods are applied by the platform. [GOOG20] In general, all proprietary systems provide an intuitive graphical user interface making it easy to understand the results and ML pipeline.

Fully comprehensive AutoML: auto sklearn 2.0 by Feurer et al. (2021)

Auto sklearn is built upon the scikit learn library and extends *Auto-WEKA* by introducing further features for addressing the CASH problem and DPP methods (Chapter 2.5). In contrast to Auto-WEKA, *auto sklearn* uses historically performant portfolios of ML pipelines to warm start the optimization process. For optimizing hyperparameters, *sequential model-based algorithm configuration* (*SMAC*) with the surrogate model *random forest* is applied (see Chapter 2.3.2). The training process is accelerated by using successive halving to shift the budget to better performing configurations. *Extra trees*, *gradient boosting*, *MLP*, *passive aggressive*, *SGD*, and *random forest* are included as ML algorithms. Uncertainty is partially addressed through averaging the outputs of several ML pipelines. [FEUR22]

From *auto sklearn* 0.6, which started only with a small set, the number of DPP methods has been increased in latest releases underlining the high impact of DPP to ML model performance. *Auto sklearn* 2.0 comprises DPP techniques of imputation, encoding,

scaling, quantile transformer, and balancing. However, for balancing, only weighted or none can be chosen. In addition, augmentation, outlier handling, and reduction methods are missing. Response latencies of *auto sklearn* can be very high due to the large search space. Even though being applicable, a high effort is required when implementing regression algorithms. Ultimately, DPP pipelines are not recommended, but the entire ML pipeline is automatically applied.

Fully comprehensive AutoML: TPOT by Olson & Moore (2016)

The *tree-based pipeline optimization tool* (*TPOT*) for automating ML uses genetic programming to identify best ML pipelines consisting of DPP and ML algorithms. Starting with simple pipelines, more complexity is introduced with increasing generations. In the beginning, 100 pipelines are constructed and subsequently 20 pipelines are selected, while maximizing the performance and minimizing the number of operators within the pipeline. Each pipeline then generates five copies based on which 10 % is crossed-over and 90 % randomly adjusted. This procedure is stopped after 100 generations. ML algorithms that are covered in *TPOT* are *decision tree*, *random forest*, *XGBoost*, *logistic regression* and *K-NN*. Olson & Moore distinguish DPP techniques into feature preprocessing and feature selection operators. Feature preprocessing consists of feature scaling, *PCA*, *binarizer* and *polynomial features*, while feature selection contains the methods *variance threshold*, *select k best*, *select percentile*, *select Fwe* and *RFE*. [OLSO16b], [LETR20] However, balancing and augmentation techniques are not covered.

Further open source AutoML approaches can be found in Annex A.5. Even though AutoML tools increasingly incorporate DPP methods, they are generally missing performant DPP methods. This shortcoming is addressed by AutoML-based DPP systems, which are introduced in the following.

AutoML-based DPP: Effective DPP for AutoML by Giovanelli et al. (2021)

Inspired by *PRESISTANT*, Giovanelli et al. combine well performing DPP methods into DPP pipelines. Based on the CASH problem, the authors introduce the data pipeline and optimization (DPSO) problem, in which most performing sequences of DPP methods are to be found before applying classification algorithms. Given a so called pipeline prototype, which is a set of DPP methods in a fixed order, a pipeline is searched that represents a sub set of the pipeline prototype. The combinatorial problem is addressed by evaluating pairs of DPP methods, which perform well on a given data set and deriving heuristic and learned rules to optimize the search for future data sets. For rule learning, *sequential model based optimization* (*SMBO*) is used as the optimization technique, which has proven to be efficient and performant in AutoML systems. As in *PRESISTANT*, imputation, encoding, feature scaling, discretization and

feature selection methods are covered. Moreover, balancing techniques are also considered in the construction of pipelines leading to a large search space when finding suitable DPP pipelines in real world applications. [GIOV21]

The PipeConstructor is updatable, extendable, and retrainable due to the SMBO approach. It further proposes a sequence of methods and partially handles uncertainty through averaging model outputs in the development phase. Although avoiding the combinatorial problem by considering pairs of DPP methods, the search space is high dimensional when considering every DPP method in the pipeline prototype. Therefore, a low response latency of the system cannot be expected. The approach lacks decisive DPP methods such as augmentation methods, while only focusing on classification tasks.

AutoML-based DPP: Alpine Meadow by SHANG ET AL. (2019)

Without explicitly naming them, SHANG ET AL. also seized the idea of combining DPSO and CASH problem to democratize the generation of performing ML pipelines by depicting the methodology of data scientists in building ML pipelines. Data scientists generally start with the implementation of baseline DPP methods and baseline ML algorithms, and afterwards increase complexity by implementing more sophisticated algorithms, if needed. Inspired by this methodology, *Alpine Meadow* automatically searches for the best possible pipeline while presenting best pipelines to the end user in periodical steps. To automatically find best performing pipelines, SHANG ET AL. decided to combine a *multi-armed bandit* and *BO approach*, whose principle has been introduced in Chapter 2.3.2. Based on a search space of possible DPP methods, the best *logical pipeline* is determined. A logical pipeline represents the best pipeline consisting of DPP methods in an unfixed order. Since the search space is very large, the *logical pipeline* is searched based on a *multi-armed bandit approach*. Once the best *logical pipeline* is found, the hyperparameters of selected DPP methods are optimized based on Bayesian optimization, more specifically *SMBO*. Ultimately, best DPP pipelines including ML algorithms are automatically calculated based on 220 classification and 80 regression data sets. [SHAN19]

Alpine Meadow focuses on democratizing data science and shows the user intermediate results. However, the interpretability of *Alpine Meadow* through the combination of multi-armed bandit and *SMBO* is thereby limited and requires computational resources, which are often not existing in the production environment. As of now, DPP categories such as power transformers, balancing, or augmentation are not covered within *Alpine Meadow*. In addition, building only 66 logical pipelines for classification took over four hours.

3.3 Assessment of Existing Approaches and Required Action

Based on the shortcomings of existing approaches, the required action is derived. For that reason, existing approaches are assessed by presenting the degree of fulfillment of derived requirements. In the following, it is assessed how corresponding approaches presented in Chapter 3.2 fulfill the requirements stated in Chapter 3.1. In this context, existing approaches can either entirely, partially, or not fulfill the requirement at hand. The results are finally summarized in Table 3-1.

Fundamental prerequisites

In every existing approach, the *combinations of DPP methods*, which are deemed important for achieving high ML model performances, are only partially fulfilled since not every DPP method from categories and steps are included. AutoML-based DPP systems, *Learn2Clean*, *Alpine Meadow*, *Effective DPP for AutoML*, and *MetaPrep* include DPP methods which contain more than one DPP category and simultaneously combine methods into pipelines.

The requirement of *handling conditionality* is addressed by the vast majority of approaches. Basic support such as cheat sheets and flow charts do not explicitly underline conditionality between different DPP methods. In case of identifying *best performing DPP pipelines* for ML applications before developing a system, AutoML systems and user assistance approaches combine DPP and ML application within the optimization process, thus finding performing DPP pipelines. *Potter's Wheel* represents one exception, which does not involve a learning process and is not tailored to ML applications. Basic approaches such as cheat sheets and flow charts do not explicitly focus on well performing pipelines. In terms of *representative production use cases* required for setting up and validating a performant learning system, the user assistance approaches of *Alpine Meadow*, *PRESISTANT*, *RankML*, *MetaPrep*, and *Effective DPP for AutoML* are validated based on at least 60 data sets. Reinforcement learning-based *Learn2Clean* or probabilistic inference-based *HoloClean* are validated on data sets in a range of three to four data sets, respectively. The requirements is fulfilled, if more than 100 data sets are used for validating, and not fulfilled if less than 15 data sets are used. However, the data sets do not solely stem from production and its particular requirements discussed in Chapter 2.1 and Chapter 2.3.2.

Basic tools that support in preprocessing data do not cover conditionalities and are not built on representative use cases. The other technologies offer the opportunity to meet the prerequisites of a DSS.

Functional requirements

One key requirement represents *complexity reduction* to ensure simple systems that recommend DPP pipelines. Cheat sheets and flow charts exhibit a reduced complexity. Open source AutoML systems as well as *Learn2Clean* and *Alpine Meadow* represent highly complex systems to identify suitable DPP pipelines. For instance, *auto sklearn* provides multiple ensembles with different weighting that can overwhelm the user of an application. *RankML*, *MetaPrep*, and *PRESISTANT* propose simpler meta learning systems with less complexity. In case of *MetaPrep*, nearest neighbor is used, which is not explainable and prone to unscaled data, whereas *PRESISTANT* and *RankML* use tree based algorithms such as *random forest* and *XGBoost* for suggesting DPP methods, whereas complexity could further be reduced by applying baseline algorithms.

Most of the systems handle *variability of use cases*. Since *HoloClean* considers many input categories and dependencies, other cleaning-focused approaches are assessed to partially fulfill the requirement although being applicable to variable use cases. AutoML systems and the remaining user assistance systems are capable of handling different meta feature values. Except for *RankML*, none of the existing approaches are entirely coping with *production specific requirements*, which are the applicability to both regression and classification tasks, while using F1 score and RMSE as performance metrics for achieving best performance. Although being applicable to regression, AutoML systems are tailored for classification tasks meaning that manual changes need to be performed to the code to be able to handle regression tasks. *Learn2Clean* and *MetaPrep* use accuracy, which is an inappropriate metric for imbalanced data sets in general and for production in particular. *Effective DPP for AutoML*, *PRESISTANT*, and *BoostClean* only focus on classification tasks.

All approaches shown in Table 3-1 also lack a *wide variety of ML algorithms*. Since DPP pipelines are highly dependent on ML algorithms, the better result is to be expected the more pipelines are constructed on different ML algorithms (see Chapter 2.4.4). While *Potter's Wheel* does not include ML algorithms, *MetaPrep* only applies *random forest* to identify suitable DPP pipelines. *Effective DPP for AutoML* considers three ML algorithms, while *PRESISTANT* applies four ML algorithms. *Learn2Clean* provides three classification, three regression, and two clustering algorithms, while *TPOT*, and therefore also *RankML*, offers six ML algorithms. Basic approaches such as cheat sheets are model-agnostic, hence, applicable for a wide range of ML algorithms.

Except for proprietary AutoML systems, all approaches state to be *updatable and extendable* by introducing further ML algorithms or further DPP methods. In case learning approaches are used to find most performant ML or DPP pipelines, *retraining capability*

is given. However, no final assessment can be made on the retraining capability since the presented approaches are recently published focusing on developing instead of retraining systems within operation. In addition, only *HoloClean* covers fully uncertainty through the probabilistic inference. Further approaches like *RankML*, *Alpine Meadow*, *PRESISTANT*, and *Effective DPP for AutoML* touch uncertainty by averaging ML model results to address the stochastic behavior. In *MetaPrep*, *Potter's Wheel*, and *BoostClean*, no information could be found how the systems handle uncertainty.

To comply with the requirements regarding the inherent functionality of the DSS, meta learning technology is suitable due to the opportunity of reducing complexity. AutoML systems, reinforcement learning, and probabilistic inference technologies are capable of partially addressing the wide variety of ML algorithms and production specificities, however, they represent complex systems to achieve performant results.

User-specific requirements

Proprietary AutoML systems provide an intuitive graphical user interface facilitating the *usability*. Based on *Learn2Clean*, BERTI-EQUILLE proposed *CleanEX* that includes the operator to comprehend the decision of *Learn2Clean*. *Potter's Wheel* offers a spreadsheet-like display for better user interaction. Therefore, *Learn2Clean* and *Potter's Wheel* partially fulfill the requirement of usability. Except for proprietary AutoML systems, other approaches like *RankML* do not provide an intuitive user interface. The missing user interface makes it difficult for production experts to comprehend and trust results.

Basic approaches do not exhibit any *response latency*, which is why cheat sheets and flowcharts entirely fulfill the requirement. AutoML systems, *Learn2Clean*, *Effective DPP for AutoML*, and *Alpine Meadow* reveal high response latency up to several hours since these systems search for best performing pipelines during the training process. *PRESISTANT* and *MetaPrep* directly recommend DPP methods and pipelines, respectively by applying the meta learning system. Thus, low response latencies are achieved, which are comparable with cleaning based approaches of *ActiveClean*, *BoostClean*, *HoloClean* and *Potter's Wheel*.

Only *MetaPrep* and *Effective DPP for AutoML* provide *ranked DPP pipeline recommendations*. *Learn2Clean*, *AlpineMeadow*, *RankML*, or AutoML systems such as *auto sklearn*, *TPOT*, and *H2O AutoML* provide rankings and weightings of pipelines, however, do not explicitly display ranked recommendations of DPP pipelines. Here, DPP pipelines can be identified within the ensembles of ML pipelines. No ranking is provided by basic approaches and user assistance that solely focus on data cleaning methods. *HoloClean* offers a high *input flexibility* by considering many different input parameters and denial constraints. For all other existing approaches, a certain input flexibility is

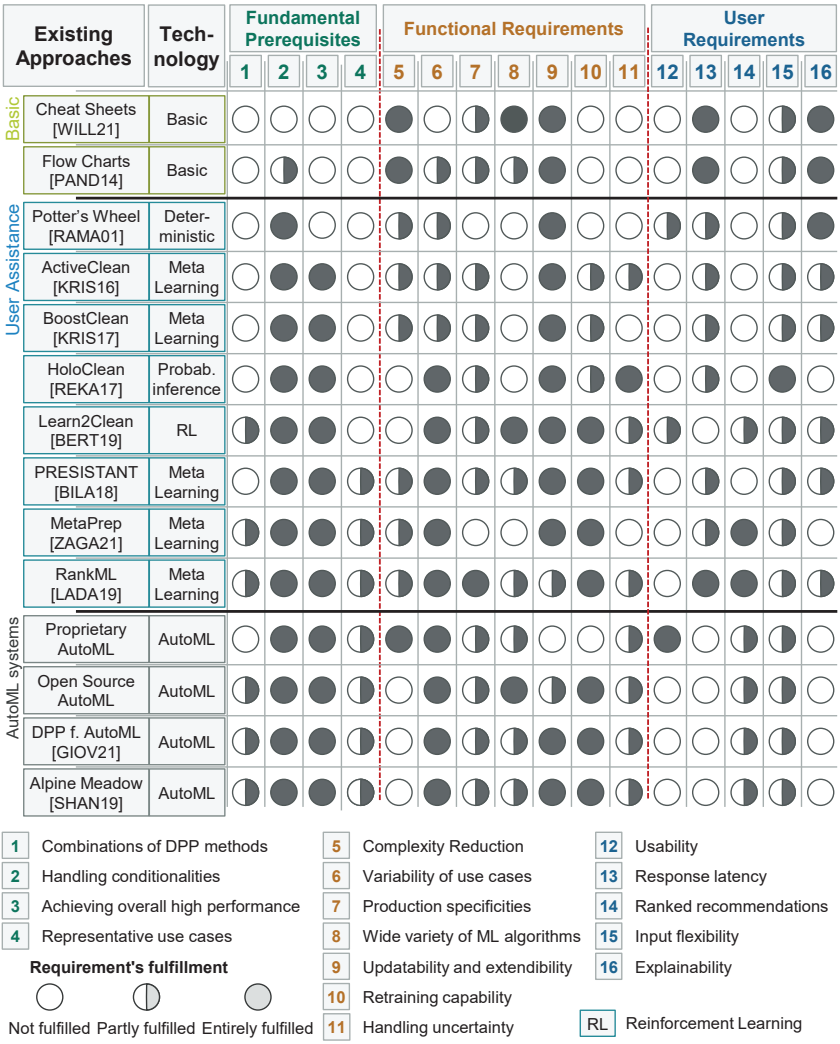
given, however, many parameters need to be inserted beforehand for system's functionality.

The *explainability* requirement is completely fulfilled for basic approaches and *Potter's Wheel*, which are easy to understand. Every AutoML system is deemed to be intransparent. In case of proprietary AutoML systems, the functionality cannot be investigated in detail, while open source systems consist of many complex calculations which is not directly traceable by the user. By utilizing SMBO, *Effective DPP for AutoML* and *Alpine Meadow* are not transparent. *MetaPrep* uses a meta learning system, which applies *K-NN* as meta model, while *K-NNs* decisions are difficult to understand. *HoloClean* applies probabilistic inference, which is barely understandable for a manual operator. Although applying Q-learning, *Learn2Clean* partially fulfills the requirement due to the extension to *CleanEX*. *PRESISTANT* applies *random forest* as meta model, whereas *BoostClean* uses *AdaBoost* and *RankML* applies *XGBoost* as learner, representing interpretable ML algorithms.

Based on the user-specific requirements, meta learning represents the most promising approach to facilitate a satisfying usability and provide ranked recommendations, while retaining transparency and low response times. As can be taken from the critical assessment, no approach entirely fulfills the requirements. Especially the small number of supported DPP methods, low variety of ML algorithms, often non-compliance with both regression and classification tasks, as well as high complexity and intransparent structure of performant approaches emphasizes the need for further research.

Ultimately, Table 3-1 shows the overview of the assessment of existing approaches. For every existing approach that is classified into basic, user assistance and AutoML systems in different colors on the left, the technology is assigned. For every existing approach, the results of the assessments of fundamental, functional and user requirements are shown. The frameworks of proprietary and open source AutoML systems are summarized. Even though the frameworks differ in terms of ML properties and underlying concepts of tuning hyperparameters, the frameworks are abstracted for DPP, since the assessment of the abstracted forms can still be ensured.

Table 3-1: Assessment of existing approaches and the fulfillment of requirements



3.4 Interim Conclusion

Starting from fundamentals, Chapter 3 presented the derivation of requirements and analysis of existing approaches followed by their assessment to deduce the need for research. Based on the established requirements in Chapter 3.1, SRQ 1 can be answered:

SRQ 1 Which requirements need to be met by the decision support system for recommending DPP pipelines?

The requirements of a DSS to recommend suitable DPP pipelines can be divided into fundamental prerequisites, inherent functionality and user-specific requirements. In total, 16 requirements were identified, which are classified into fundamental prerequisites, functional, and user-specific requirements. Existing approaches were then analyzed and critically assessed by means of previously stated requirements. In conclusion, SRQ 2 can be answered:

SRQ 2 Which approaches and technologies can serve as a basis for the development of a decision support system?

Promising approaches have been introduced, which cover partially the requirements. Basic approaches provide an initial support in identifying DPP methods but do not provide comprehensive recommendations. Fully comprehensive AutoML systems are highly performant and increasingly cover DPP methods. However, due to the focus on modelling, these AutoML systems do currently not provide enough DPP methods. AutoML-based DPP systems such as *Effective DPP for AutoML*, and *Alpine Meadow* cover various DPP methods. The high response latencies and high complexity do not fulfill the requirements. User assistance approaches can rely on different technologies such as reinforcement learning, probabilistic inference, or meta learning. *Learn2Clean* and the extension *CleanEX* applies reinforcement learning, which is highly complex and requires many iterations until convergence. Meta learning-based approaches like *PRESISTANT*, *MetaPrep*, and *RankML* find promising tradeoffs between the complexity and performance of the system. The use of meta learning can result in explainable systems, while providing recommendations in a short time. Furthermore, meta learning has especially proven to be performant in those systems. For these reasons, meta learning will be used to develop a DSS. Since this work aims at developing a DSS for recommending DPP pipelines for ML applications in production, Chapter 4 comprises the development of a DSS for DPP pipeline recommendation in production that is verified and validated in Chapter 5.

4 Development of a Meta Learning System for DPP Pipelines in Production

In this chapter, the meta learning-based DSS is introduced for recommending most suitable DPP pipelines for ML applications in production, which is called Meta-DPP. The first sub chapter comprises the general design of Meta-DPP. Subsequently, Chapter 4.2, 4.3, and 4.4 provide details about the development of three core components of Meta-DPP. In the first step, a meta target selector is developed, which aims at choosing a suitable target for the meta model (Chapter 4.2). Chapter 4.3 contains the creation of a meta features database that provides input features for the meta model. Given the meta features database and the selected meta target, a meta model is created, which recommends most suitable DPP pipelines for a given use case (Chapter 4.4). Chapter 4.5 describes the technical implementation and retraining concept to ensure long-term usage of Meta-DPP. Chapter 4 concludes with an interim conclusion (Chapter 4.6).

4.1 General Design of Meta-DPP

In the following, the general design and functionality of Meta-DPP are outlined. In a first step, fundamental design considerations of the meta-learning based system are discussed in Chapter 4.1.1. A distinction is made between applying and developing Meta-DPP. Chapter 4.1.2 describes the functionality of core components and their interactions as well as how the user applies Meta-DPP. Then, Chapter 4.1.3 provides an overview about the development of the core components, which are detailed in Chapter 4.2, 4.3, and 4.4.

4.1.1 Fundamental Design Considerations of Meta-DPP

Meta-DPP at its core is based on the concept of meta learning (see Chapter 2.5). According to Figure 4-1, a meta target (I) consists of a set of DPP pipelines to be recommended. The meta target of Meta-DPP represents a preselected set of overall well performing DPP pipelines, called pipeline pool. The pipeline pool results from the meta target tradeoff: On the one hand, many different best performing pipelines exist across production use cases resulting in a very high number of classes, i. e., DPP pipelines, in the meta target. On the other hand, a meta model requires many historical instances per class to learn a pattern directing towards a lower number of classes. This tradeoff is addressed through identifying a set of overall well performing pipelines across many production use cases, i. e., a pipeline pool, to ensure the existence of several instances per class. Since DPP pipelines further depend on the learning task, two pipeline pools

are identified for classification and regression. This concept requires an approach that is called meta target selector, which chooses between different pipeline pools based on the learning task.

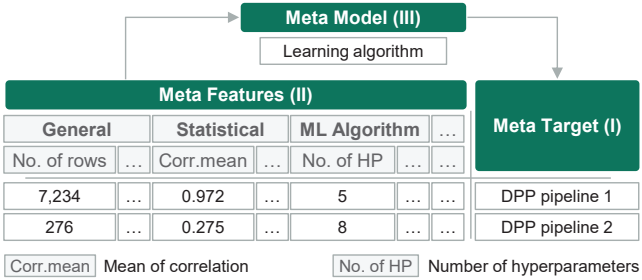


Figure 4-1: Meta learning-based concept for recommending DPP pipelines

The characterization of production use cases is realized by the extraction of meta features (II). While too many meta features lead to the curse of dimensionality as discussed in Chapter 2.4.3, too few meta features may not provide enough information. Therefore, different dimensions of meta data are considered, which extract information from the data set at hand, ML algorithm performance, and historical DPP pipeline performance. Given the selected meta target and meta features, a meta model (III), e. g., a learning algorithm, is trained that aims at predicting the best performing pipeline based on use cases' meta features. Given these fundamental considerations, the application of Meta-DPP and functionality of core components are introduced.

4.1.2 Core Components of Meta-DPP and its Interactions

The functionalities and interactions of core components are described based on how Meta-DPP is applied by a user. The application of Meta-DPP, which is a meta learning-based system for recommending DPP pipelines for ML applications in production, can be taken from Figure 4-2. Starting from the bottom left, a user, e. g., a production expert, an IT expert, or a data scientist can input their production use cases by providing the data set, selecting an ML algorithm, and specifying additional requirements such as explainability.

The input is used to apply a meta target selector, which chooses between two DPP pipeline pools dependent on the learning task of either classification or regression. For each learning task, a pool of overall well performing DPP pipelines is available. In parallel, meta features are directly extracted dependent on the production use case provided by the user. According to Chapter 2.5, given the meta target, i. e., pipeline pool, and input variables, i. e., meta features, a meta model aims at predicting the most

performant DPP pipeline from the preselected pipeline pool based on use case characteristics.

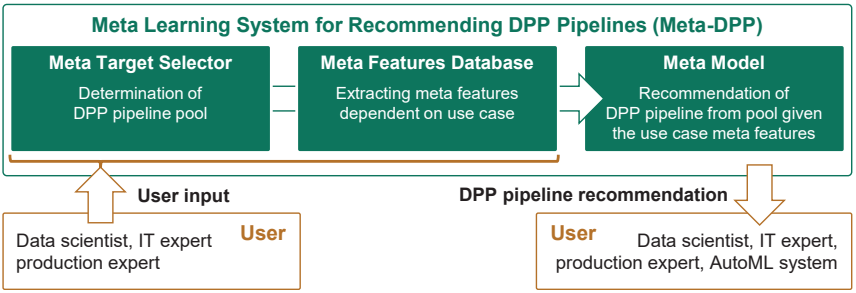


Figure 4-2: Application of Meta-DPP and its core components

The outcome of the meta model is a ranking of DPP pipelines, which can be applied by the user. In addition, the outcome can also be applied as warm start or initial solution for AutoML systems. The main components are briefly explained in the following.

Meta target selector

The meta target selector consists of a rule based system, in which the pipeline pool is chosen depending on the learning task. Based on the identification of overall best performing pipelines, the pipeline pool consists of a fixed set of DPP pipelines for each classification and regression. A thorough overview on the development of the meta target selector is provided in Chapter 4.2.

Meta features database

The meta features database stores information about production use cases and past performances of ML algorithms and DPP pipelines. To facilitate the generalization of the meta model, the meta features database consists of three meta data dimensions, which are data set-, ML algorithm-, and DPP pipeline-related. Data set-related meta features are classified into general, statistical, information-theoretical, landmarking, and model-based. General meta features provide information about data set's shape, e. g., number of instances. Statistical and information-theoretical meta features investigate the properties of features and their relationships, e. g., through correlations between features. Landmarking trains simplest forms of baseline models, e. g., a decision node of a *decision tree*, on data sets to describe the task of recommending DPP pipelines. In case of model-based meta features, model parameter values of a trained *decision tree* are related to the characteristics of a data set, e. g., the maximum depth related to the number of attributes.

Further meta features stem from the properties as well as past performances of ML algorithms. Properties of the ML algorithms are, for instance, the number of hyperparameters. DPP pipeline-related meta features are further calculated. Best performing DPP pipelines are used to count the occurrences of DPP methods in best performing pipelines. Best performing DPP methods are then assigned to DPP categories and DPP steps as described in Chapter 2.4.3. Since the meta features depend on the learning task, two different sets of meta features are available for regression and classification. The details about the development of meta features database are provided in Chapter 4.3.

Meta model

Since two different pipeline pools and sets of meta features exist due to the learning task distinction, two meta models are used to recommend DPP pipelines. The meta model outputs a ranked recommendation by providing probabilities of each class, i. e., DPP pipeline. Probabilities are used in two ways. First, the user receives a ranked recommendation, from which DPP pipelines can be implemented according to the order of probabilities. Second, probabilities give insights about how certain a model is about the recommendation.

Based on how Meta-DPP is applied, a focus is put on the development of Meta-DPP in the following. For that reason, Chapter 4.1.3 shows an overview on how the core components are created.

4.1.3 Overview of Developing the Core Components of Meta-DPP

Following up on Figure 4-2, which describes the functional sequence when using Meta-DPP, Figure 4-3 shows the development workflow of Meta-DPP in blue boxes. First, the meta target selector is developed, which chooses the pool of pipelines dependent on the learning task. Representative production use cases are identified and suitable DPP pipelines configured, which are benchmarked according to the performance of ML algorithms. The benchmarking results are ranked in the following to determine DPP pipelines, which perform well over many production use cases. Ultimately, the pipeline pool with a fixed set of DPP pipelines per learning task is identified (Chapter 4.2).

The meta features database is constructed based on the determination of relevant data set-, ML algorithm-, and DPP pipeline-related meta features. The specificities of the meta features database development can be taken from Chapter 4.3. Based on the developed meta features database and identified pipeline pools, the two meta models are created in Chapter 4.4. In a first step, a suitable meta pipeline consisting of DPP pipeline and ML algorithm is selected. Secondly, the meta models are implemented to recommend DPP pipelines.

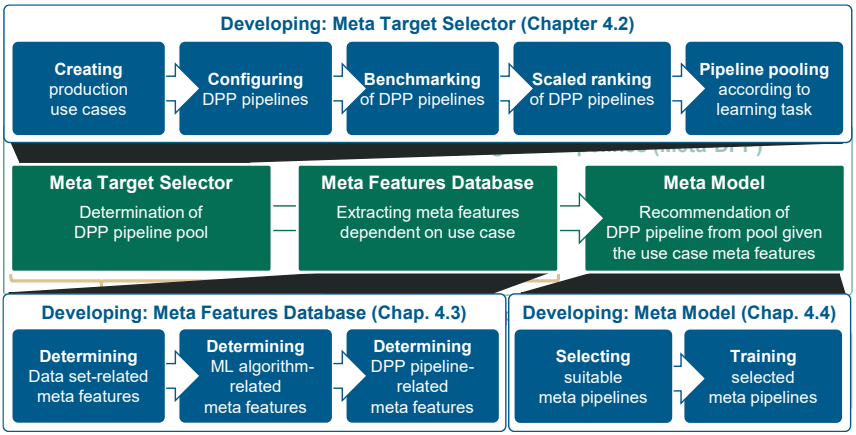


Figure 4-3: Development process for the core components of Meta-DPP (depicted in blue)

4.2 Developing the Meta Target Selector

The goal of the meta target selector is to choose a preselected pool of overall well performing DPP pipelines dependent on the learning task. To set up the meta target selector, in the first step, representative production use cases are determined, which build the basis for finding best performing DPP pipelines (Chapter 4.2.1). Based on a selection of suitable DPP methods, DPP pipelines are configured in Chapter 4.2.2. Through the benchmarking of previously combined DPP pipelines on representative production use cases, the performance of DPP pipelines is determined according to ML model scores (Chapter 4.2.3). The outputs are subsequently ranked in a scaled manner. Over all use cases and DPP pipelines, descriptive statistics are derived by calculating the mean, median, minimum, maximum, and various percentiles (Chapter 4.2.4). The results are used to eventually find two pools of DPP pipelines dependent on the learning task in Chapter 4.2.5: one for regression and one for classification. These pools represent classes of the meta target. Figure 4-4 highlights the steps for developing the meta target selector and refers to the chapters, in which the sub components are discussed.

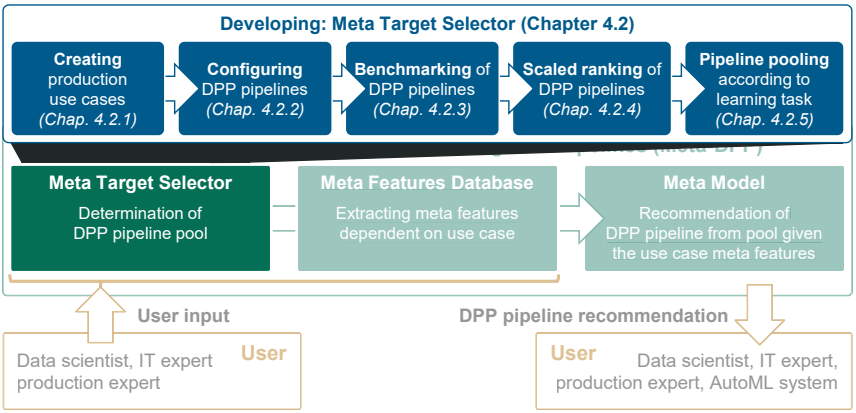


Figure 4-4: Procedure of developing the meta target selector

4.2.1 Creating Production Use Cases

A production use case is the input that a user provides to Meta-DPP to receive a recommendation of DPP pipelines. The user can input a data set, ML algorithm, and the requirement of the availability of explainable methods. Since explainability will be used to filter DPP pipelines in the later stage, production use cases for developing the meta target selector are built on data sets and ML algorithms.

Identification of suitable production data sets

Since numerous production use cases should serve as the basis for benchmarking DPP pipelines, 44 publicly available, tabular data sets related to production and six data sets from projects conducted at Fraunhofer IPT are used. The 44 production use cases are selected based on detailed research, which can be taken from KRAUß ET AL. [KRAU19b]. In total, 50 data sets are considered. Particular focus is put on the availability of representative production use cases from the application areas introduced in Chapter 2.1. Therefore, 32 % of the data sets are related to predictive process control, 26 % to predictive maintenance, and 14 % to process design. Further data sets are assigned to the applications of anomaly detection, product design, and process management. For every data set, meta features, data set descriptions and references can be taken from Annex A.6.

As described in Chapter 2.3.3, the performance of ML algorithms highly depend on the existence of missing values and categorical columns. In addition, the recommendation of fully comprehensive DPP pipelines rely on imputing and encoding strategies. Since

many data sets do not contain missing values or categorical columns and to simultaneously increase the number of production use cases, derivative data sets are created from existing ones to achieve a balance of data sets with:

- no missing values and no categorical columns (*norm*),
- no missing values and categorical columns (*cats*),
- missing values and no categorical columns (*mvs*),
- missing values and categorical columns (*both*).

The *norm* data sets with only numerical columns are used to randomly introduce categorical features. Numerical columns are randomly picked and discretized through binning using equal width. In addition, missing values are randomly introduced to *norm* data sets. If both categorical and missing values are introduced, “both” data sets are created. Ultimately, 104 data sets are used for benchmarking, while 75 data sets are classification and 29 data sets regression tasks. To gain insights of the chosen data sets, Table 4-1 provides an excerpt of the data sets being used. For every data set, the learning task (*LT*), number of instances (*Inst.*), number of attributes (*Attr.*), number of categorical columns (*Cats*) and number of missing values (*MV*) are shown.

Table 4-1: Overview of data sets (excerpt)

No.	Data Set	LT	Inst.	Attr.	Cats	MV	CL / STD	Ref.
01	3D Printer	RE	50	11	2	0	0.7802	OKUD19
02	SECOM	CL	1,567	591	0	41,951	1463:104	DUA19
03	SFGD	CL	2,109	21	0	0	1783:326	SAYY05
04	SFGD - mvs	CL	2,109	21	0	2,214	1783:326	SAYY05
05	SFGD - cats	CL	2,109	21	9	0	1783:326	SAYY05
06	SFGD - both	CL	2,109	21	4	1,772	1783:326	SAYY05
07	Turbofan 1	RE	13,096	26	0	0	41.39	SAXE08
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
104	Valve mvs	CL	2,205	38	0	2,253	360:360: 360:1125	HELW15

LT	Learning task	Ref.	Reference	SFGD	Software for ground data
Inst.	No. of instances	MV	No. of missing values	CL	Classification
Attr.	No. of attributes	Cats	No. of categorical columns	RE	Regression
CL / STD	CL (classification): ratio of classes STD: standard deviation of label (regression)				

In case of classification tasks, the class representation is recorded, while for regression, the standard deviation of the target variable is presented (CL/STD). Finally, the reference (Ref.) of the publicly available data set is given. The entire list of 104 data

sets can be taken from Annex A.6. While Table 4-1 provides an excerpt of the data sets being used, Table 4-2 shows the descriptive statistics of the data sets. Thereby, 32 data sets of type *norm* neither contain missing values nor categorical columns, 30 data sets exhibit categorical columns, 22 data sets *MVs*, and for 20 data sets, both *cats* and *MVs* are present. The number of rows ranges from 40 to 600,000, while the number of columns lies between 5 and 591. The maximum number of *cats* is 45, while the maximum number of MVs is 41,951. ML algorithms further create production use cases, which are introduced in the following.

Table 4-2: Descriptive statistics of data sets

No. of data sets	No.
Norm	32
With cats	30
With mvs	22
With both	20
RE/CL Ratio	29/75

No. of…	Min	Max	Mean	Median
rows	40	76,000	6,689.1	2,205
columns	5	591	60.81	21
cats	0	20	2.5	0
MVs	0	1,078,695	14,616.1	0

Identification of suitable ML algorithms

According to Chapter 2.3.3, ML algorithms and DPP are highly interrelated. Therefore, ML algorithms for both regression and classification are considered as input dimension for Meta-DPP. To provide the user of Meta-DPP a broad applicability, 18 ML algorithms are selected given the following criteria, which are formulated based on an extensive literature research:

- Successful application in production (see Chapter 2.1)
- Tabular data (see Chapter 2.2)
- Supervised learning (see Chapter 2.3.2)
- Mix of baseline and more complex ML algorithms (Chapter 2.3.3)
- Interpretable and non-interpretable ML algorithms (Chapter 2.3.3)
- Different levels of robustness against data quality issues (Chapter 2.3.3)
- Computational complexity (Chapter 2.3.3)
- Ease of implementation

By applying these criteria, Table 4-3 shows the overview of 18 regression and classification algorithms, which exhibit input options for the user of Meta-DPP and which are used to benchmark DPP pipelines. In addition to the considerations of Chapter 2.3.3, *bagging* is performed with the baseline model *K-NN*, since *K-NNs* provide reasonable performance in the presence of scaled data, while being fast. *Voting* is based on two competing baseline models, which are *DT* and *K-NN*, since *DT* and *K-NN* complement

each other in terms of properties. For the first layer of the *stacking* algorithm, *DT* and *K-NN* are applied. The final model in the second layer is represented by *logistic regression* in case of classification, and *linear regression*. The learning approach of *SGD* introduced in Chapter 2.3.2 is used with a *linear-kernel SVM* representing a fast yet performant ML algorithm. In total, 18 ML algorithms can be selected in Meta-DPP.

Table 4-3: ML algorithms for classification and regression considered in Meta-DPP

No.	Classification	Regression
01	Logistic regression	Linear regression
02	Ridge classifier	Ridge regressor
03	Naïve Bayes classifier	Elastic net
04	Gaussian process classifier	Gaussian process regressor
05	K nearest neighbor classifier	K nearest neighbor regressor
06	Support vector classifier	Support vector regressor
07	Decision tree classifier	Decision tree regressor
08	Random forest classifier	Random forest regressor
09	Extra trees classifier	Extra trees regressor
10	AdaBoost classifier	AdaBoost regressor
11	Gradient boosting classifier	Gradient boosting regressor
12	LGBM classifier	LGBM regressor
13	Histogram gradient boosting classifier	Histogram gradient boosting regressor
14	Bagging (K-NN)	Bagging (K-NN)
15	Stacking (DT, K-NN, logistic regression)	Stacking (DT, K-NN, linear regression)
16	Voting (DT, K-NN)	Voting (DT, K-NN)
17	Multilayer perceptron classifier	Multilayer perceptron regressor
18	SGD classifier	SGD regressor

In total, 104 production data sets and 18 ML algorithms yield *1,872 production use cases* applicable for benchmarking DPP pipelines and developing the meta model in Chapter 4.4. Before benchmarking, DPP pipelines need to be configured based on suitable DPP methods. The configuration of suitable DPP pipelines is outlined in the following subchapter.

4.2.2 Configuring Suitable DPP Pipelines

The meta target selector relies on a pool of DPP pipelines dependent on the learning task. To configure suitable pipelines, DPP methods need to be determined based on predefined criteria. For that reason, DPP methods are first selected (A) and subsequently configured into DPP pipelines (B).

A) Selection of suitable DPP methods

The identification of suitable DPP methods plays a fundamental role to ensure a performance increase of ML applications. Given the frame of DPP steps and categories presented in Chapter 2.4.3, most suitable DPP methods are selected in the following. As a first step, decision criteria are introduced to preselect performant DPP methods and to configure a manageable amount of DPP pipelines. If the application of decision criteria lead to remaining, similar DPP methods, individual benchmarks are conducted in a second step to determine most suitable DPP methods.

Definition of decision criteria

The definition of decision criteria and the subsequent assessment of DPP methods is based on an extensive literature research. Based on identified overall DPP methods for preparing tabular data sets in Chapter 2.4.3 and Annex A.1-A.4, DPP methods are preselected applying decision criteria. Different tradeoffs need to be considered when selecting DPP methods. In the following, eight decision criteria including its assessment metrics are introduced. The order represents the priority of the criteria. For every decision criterion, “1” represents complete and “0” no fulfillment, while the assessment proceeds in quarter steps, i. e., “0.00, 0.25, 0.50, 0.75, 1.00”. The assessment is to be seen relatively, i. e., related to the respective DPP category.

1. **Overall performance**: DPP methods need to be well performing on use cases with different characteristics. The goal is to determine overall performant DPP methods across all use cases. If the DPP method is overall performant, “1” is assigned, if the performance is poor, “0” is entered.
2. **Fast computation:** DPP methods are chosen that perform as fast as possible to reduce execution time. In case DPP methods are equally performant, faster methods are selected. Very fast DPP methods receive “1”, computationally expensive DPP methods receive “0”.
3. **Complexity & explainability**: DPP methods should be easy to comprehend. If DPP methods perform equally well, simple and explainable DPP methods are preferred to reduce the complexity of the DPP pipeline. Very simple and explainable DPP methods are assigned “1”, very complex and unexplainable ones “0”.
4. **Impact**: Although there are versatile DPP methods with different effects, DPP methods can introduce bias and noise, which manipulates the data set. The degree of manipulation can be measured through the difference in the shape of the data set and variance of features after applying the DPP method. DPP methods are selected, which exhibit a reasonable impact on the data set to retain

most of the data set. DPP methods, which have a high impact, get “0”, and DPP methods with low impact receive “1”.

5. **Robustness**: DPP methods are chosen that are robust against data set’s noise. Therefore, stochastic behavior, or randomness, of DPP methods should be in a manageable range. The stochastic behavior is measured based on the number of random parameters of the DPP method. If two DPP methods are equally well performing, deterministic DPP methods are selected. Deterministic DPP methods therefore receive “1” being repeatable independently from data input, stochastic DPP methods get “0”.
6. **Popularity & stability**: If DPP methods are frequently applied in literature, a DPP method is selected. In addition, DPP methods need to be commonly used and stable, i. e., represented and thoroughly tested by a wide community, while existing in common libraries that can be used in production. If the theoretical background of a DPP method is promising but no adequate library or stable version exists, DPP methods are disregarded. If a DPP method is popular and stable, it is assigned a “1”.
7. **Ease of implementation:** To facilitate the application of DPP methods by the user, easy to implement DPP methods are selected. Easy to implement are methods, that can be called through a corresponding library. DPP methods being very easy to implement get “1”.
8. **Heterogeneity of DPP methods within category:** Another objective is to select heterogeneous DPP methods for covering as many data set characteristics, and ML algorithm requirements as possible. DPP methods are selected within a DPP category, which show heterogeneity.

Besides the decision criteria, conditionality of different DPP methods needs to be considered. If performant DPP methods require other DPP methods for functionality, corresponding DPP methods are included. When selecting, the conditionality of the DPP methods are investigated and considered.

For every DPP method mentioned in Annex A.1-A.4, the first seven decision criteria are applied in an initiating step. For the seven criteria, the mean is calculated based on the assessment scores, which lies between “0” and “1”. DPP methods that achieve the highest assessment score are initially selected. To ensure heterogeneity between DPP methods, the eighth decision criteria is subsequently applied. In case preselected DPP methods are still similar in nature after applying the last decision criterion, an individual benchmarking at small scale, which is based on Frye et al. and detailed in this work as described in the following paragraph [FRYE21a].

Individual benchmarking at small scale

To make a profound decision, ten representative data sets are chosen from the list of 104 data sets. For every data set, the derivatives of *mvs*, *cats* and *both* are used to benchmark the performance of MV handling methods and encoders. Annex A.6 provides the ten selected data sets. For individual benchmarking, the algorithms *RF*, *SVM* and *MLP* are trained, while F1 scores and RMSE values are acquired. Given the structure of the DPP steps data cleaning, transformation, reduction, as well as augmentation and balancing, DPP methods are selected with respect to the DPP category. When selecting DPP methods and conducting individual benchmarking at small scale, implementations from scikit learn and pandas are used, which represent two most commonly used libraries in the data science community. The following assessment is drawn from the set of DPP methods discussed in Chapter 2.4.3.

Data cleaning

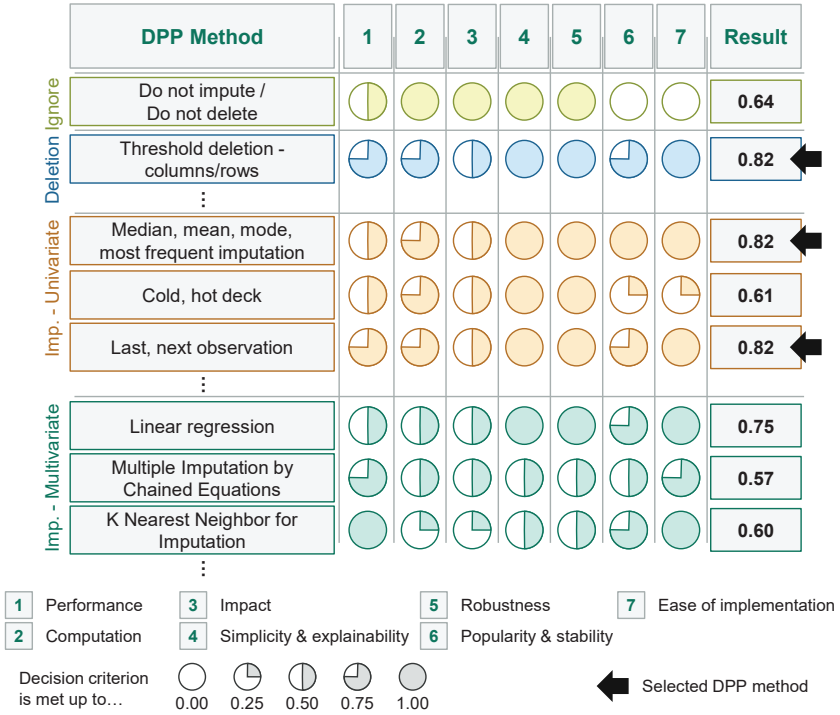
Data cleaning can be distinguished in handling missing values, outliers, and noisy data. Applying decision criteria to the DPP category *MV handling* results in the overview shown in Table 4-4. For better readability of the assessment according to previously defined decision criteria, Harvey balls are used to distinguish between different scores. The result, i. e., mean of assigned scores, is depicted on the right. Classifying MV handling into ignore, deletion, univariate and multivariate imputation is realized by using different color schemes.

When comparing DPP methods *median, mean, mode, and most frequent imputation* with *K-NN imputer* (*K-NNI*), *K-NNI* exhibits a higher performance. Computing times for median imputation are very low, therefore, the score of 0.75 is assigned, whereby only ignoring is faster, which would mean the achievement of a score of 1. Since the *median imputer* is explainable, deterministic, very popular, and easy to implement, the method receives the score of 1.0 ending up in a mean score of 0.84. This means that the DPP method fulfills the criteria by 84 %. Every DPP method is assessed in this manner, whose results are shown in Table 4-4 on the right. The entire assessment can be taken from Annex A.7. Consequently, six DPP methods are selected from the list of 31 DPP methods for MV handling, namely:

- Threshold deletion, or *missing value ratio*
- *Median imputation*
- *Mean imputation*
- *Mode imputation* or *most frequent imputation* in case of MVs in categorical columns
- *Last observation imputation*
- *Next observation imputation*

The eighth criteria, i. e., the heterogeneity of DPP methods within category, splits the DPP methods into threshold deletion, or MV ratio, and the imputation techniques. To narrow down the number of imputation techniques, an individual benchmarking at small scale is required.

Table 4-4: Excerpt of assessment scores for determining suitable DPP methods for MV handling (data cleaning)



Therefore, the ten data sets from Annex A.6 with corresponding derivatives of *mvs* are used for both classification and regression. The results of individual benchmarking of remaining MV handling methods are depicted in Table 4-5. F1 scores and RMSE values are provided including the computing time. *MedianImputer*, *MeanImputer* and *MostFrequentImputer* represent similar performance for both regression and classification. While the *MedianImputer* achieves better overall performance, the *MeanImputer* is the fastest performer. The *LastObservationImputer* exhibits the highest computing time, which is almost three times higher than the *MeanImputer*. However, *last observation* generally performs best, which is why *last observation* is selected over

next observation. Since MV handling is deemed as decisive step within DPP, four imputing techniques are chosen.

Table 4-5: Results of individual benchmarking for remaining MV handling methods

		CL - F1 Score			RE - RMSE			
	DPP Method	RF	SVM	MLP	RF	SVM	MLP	CT [s]
Imputation - Univariate	Mean	0.891	0.857	0.761	3.287	5.725	7.804	0.063 ←
	Median	0.888	0.859	0.768	3.142	5.771	7.410	0.135 ←
	Most frequent	0.887	0.858	0.767	3.128	5.762	7.183	0.126 ←
	Next observation	0.892	0.862	0.739	2.852	5.700	10.15	0.157
	Last observation	0.900	0.862	0.761	2.796	5.677	7.734	0.165 ←

1.000 Best performance 0.000 Worst performance ← Selected DPP method

For these reasons, both *MeanImputer* and *MedianImputer* are selected. In case of categorical columns, *MostFrequentImputer* is used. Ultimately, the following DPP methods for MV handling are selected:

- Deletion: *Threshold deletion*, or *MV ratio* (threshold = 1.0)
- Univariate imputation:
  - *MeanImputer*, *MedianImputer*, *LastObservationImputer* for numerical,
  - *MostFrequentImputer* in case of categorical columns

In the following, a short overview is given for each DPP step and corresponding categories, which DPP methods are selected based on the application of decision criteria. According to Chapter 2.4.3, there are 14 outlier handling methods to choose from. When applying decision criteria, *z score* is selected to identify outliers for every column. Subsequent to determining, outliers are imputed by the mean of the attribute (*ZSMean*). Considering the decision criteria, DPP methods of removal of duplicated rows and columns are selected. Since noise in the data has a crucial impact on the final outcome, features with purely constant as well as zero values are removed. Assessment scores for noisy data and outlier handling can be taken from Annex A.7. Ultimately, DPP methods selected within data cleaning are shown in Figure 4-5.

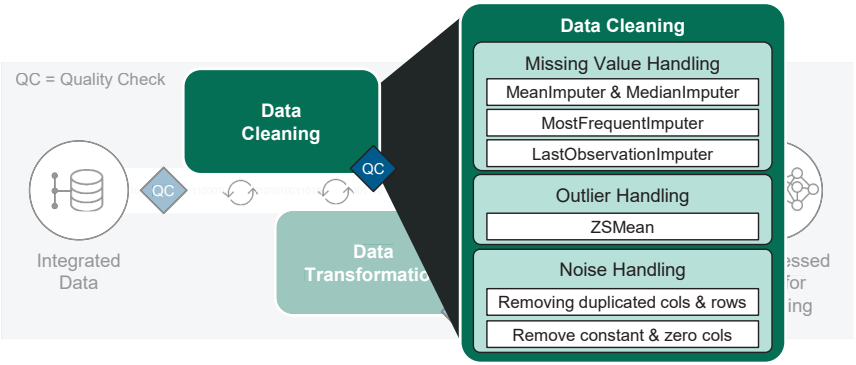


Figure 4-5: Selected DPP methods for data cleaning

Data transformation

Chapter 2.4.3 classifies data transformation in encoding, feature scaling, handling skewness, and data discretization. For encoding, classic encoders of *OrdinalEncoder*, and *OneHotEncoder* are used to configure suitable DPP pipelines. In addition, the *TargetEncoder* is chosen based on the application of the eighth decision criteria. In case of feature scaling, the *robust scaler* is omitted since outlier handling is carried out in data cleaning. Standardization and normalization are considered. Standardization is named *StandardScaler*, while normalization is realized by a *MinMaxScaler*. Due to the handling of positively and negatively skewed data, the power transformer *Yeo-Johnson* is considered, called *PowerTransformer*. Due to the high complexity in production use cases, the event of discretizing data, i. e., the reduction of the number of instances, happens rarely. Therefore, data discretization like binning is not taken into account when configuring DPP pipelines. Figure 4-6 shows the selected DPP methods for transforming data. The entire assessments for data transformation and hyperparameters of selected DPP methods can be taken from Annex A.8.

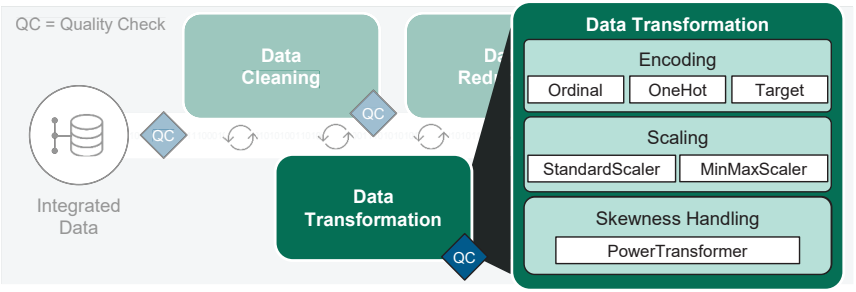


Figure 4-6: Selected DPP methods for data transformation

Data reduction

Data reduction can be distinguished between dimensionality reduction, feature selection, and instance selection. *PCA* is selected as it represents a simple yet powerful DPP method for dimensionality reduction. When applying the decision criteria, the feature selection methods, *low variance filter*, called *VarianceThreshold*, and *high correlation filter*, named *RemoveCorrelated* achieve highest scores. Besides filtering, wrapper approaches are used to reduce the number of features. While *BFE* and *FFS* represent very similar approaches, *RFE* stand out from *BFE* and *FFS* through much faster computing times while being comparably performant. Embedded methods are not taken into account, since a focus is put on DPP solely. The entire results of the assessment and hyperparameter settings of selected DPP methods can be seen in Annex A.9. Due to the fact that instance selection is nowadays performed manually, and proposed filter and wrapper methods are neither commonly used, nor easy to implement, the selection of instances is not included. Figure 4-7 represents the selected DPP methods for data reduction.

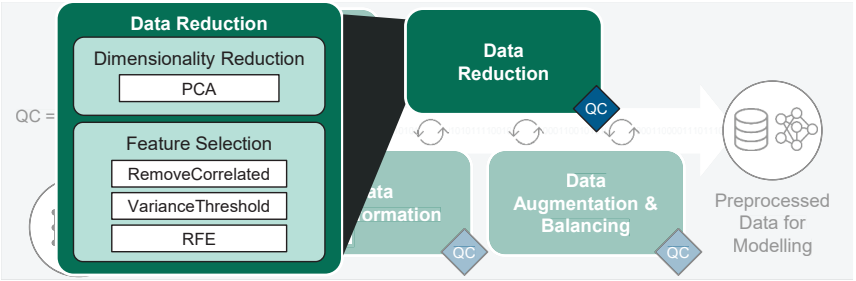


Figure 4-7: Selected DPP methods for data reduction

Data augmentation & balancing

Data is augmented through *random noise*, called *RandNoise*. In terms of balancing, *SMOTE* is used for oversampling the data. In case a small set of categorical features is available in the data set, *SMOTENC* is selected, since *SMOTENC* is tailored to the treatment of categorical features. In the remainder of this work, the distinction between *SMOTENC* and *SMOTE* is omitted, while reference is always made to *SMOTE* for better readability. For undersampling the data, the random undersampler (*RandomUndersampler*) is selected. The application of decision criteria results in a draw for the hybrid sampling methods, namely *SMOTEENN* and *SMOTETomek*. Since the DPP methods are similar, an individual benchmarking is carried out, in which *SMOTEENN* achieves higher F1 scores. The results from the benchmarking at small scale, entire assessment and hyperparameter settings can be found in Annex A.10.

Balancing in case of regression tasks can be carried out through *SMOGN*. The final augmentation and balancing techniques can be taken from Figure 4-8.

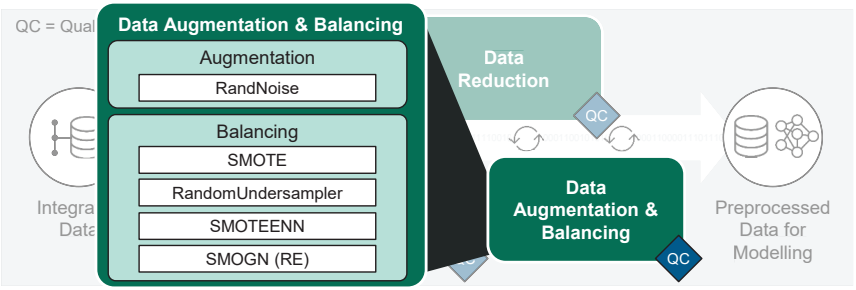


Figure 4-8: Selected DPP methods for data augmentation and balancing

In total, 21 DPP methods are used to configure DPP pipelines that are benchmarked. This provides the basis to find best performing pipelines, which are grouped in a pool of pipelines and serve as meta target for the meta model.

B) Configuration of suitable DPP pipelines

In practice, DPP methods are combined to DPP pipelines in very different orders. Only meaningful DPP pipelines yield to ML model performance increase. For Meta-DPP, as a starting point for every DPP pipeline, deterministic DPP methods are called that address fundamental noise in the data set. This includes duplicates in rows and columns, columns with constant and only zero values. These quality issues in the data set are eliminated by removing duplicated rows and columns, constant as well as zero columns. These four DPP methods are called essential methods since they represent an essential part of every DPP pipeline. In case MVs and categorical features are present, the pipeline, which additionally contains imputer and encoder, are also stated as essential pipelines. After applying essential DPP methods in advance of each pipeline, one DPP method of each category is selected once per DPP pipeline. In total, a maximum of eleven DPP methods per pipeline can be applied.

When combining DPP methods, conditionalities are considered. As described, *MinMaxScaler* or *StandardScaler* are applied prior to *PCA* as well as *VarianceThreshold* (see Chapter 2.4.4). The methods for scaling, handling skewness and dimensionality reduction are tailored for numerical data instead of categorical data. Thus, categorical features are put aside once encoded and brought back when augmenting and balancing the data. The order of possible configurations of DPP methods into pipelines is leaned on the order of DPP steps. As a first step, MVs are handled representing one of the most basic data quality issue, since many ML algorithms do not work in the

presence of MVs. In case of categorical features, MVs are handled by *MostFrequentImputer*. To ensure ML algorithm's functionality, data is encoded in case of the occurrence of categorical features in the data set. Encoded features are subsequently put aside. Outliers are then detected and handled using the *MeanImputer*. Thereby, outlier handling is carried out before outlier-prone approaches such as scaling. Numerical features are followingly scaled, which exhibits a necessary condition for the application of *PCA* and *VarianceThreshold*. According to the DPP steps and categories order, features are scaled, and transformed by applying *Yeo-Johnson* (*PowerTransformer*). Since *Yeo-Johnson* power transformer does not impact feature variances or correlations, it can be applied prior to data reduction methods.

After data transformation, *PCA* and feature selection are applied. The reduction of columns is performed prior to augmenting instances, since computing time decreases in cases of lower dimensions. Before augmenting and balancing the data, encoded features are brought back to manage the number of instances in the same manner as numerical data. For *random noise*, the original value is retained in case of the encoded features, and by applying *SMOTE* for both oversampling and hybrid sampling, synthetic data only attains categorical values for categorical columns. Since a *RandomUndersampler* only selects and not generates data, it is both applicable for categorical and numerical features. The possible DPP pipelines that aim to be most performant, fast and applicable for every tabular data set and supervised learning algorithm can be taken from Figure 4-9. Even though not being depicted, the *MostFrequentImputer* is inherently applied in case of categorical features.

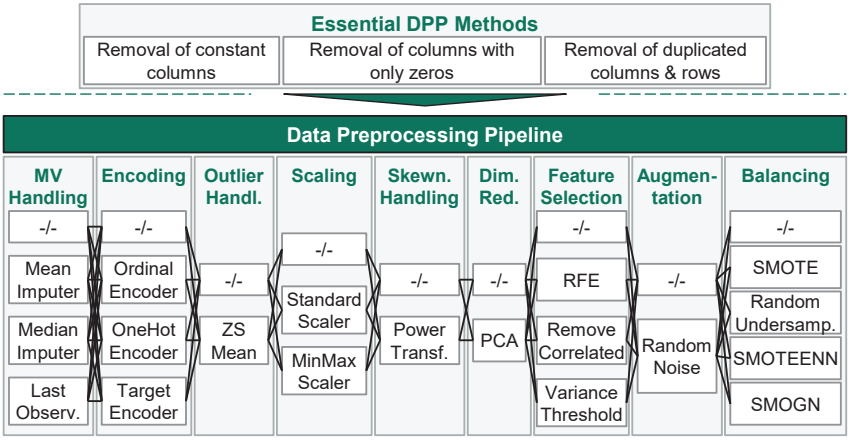


Figure 4-9: Configuration of DPP methods into DPP pipelines

In practice, not every DPP category is always required in each pipeline as indicated by the "-/-" operator. However, all possible pipelines with each possible number of methods per category are considered, which results in different pipeline lengths. Pipelines can contain only one single method of an arbitrary DPP category, e. g., *PCA*, or a combination of DPP methods from every DPP category. In total, 12,160 DPP pipelines can be constructed. Figure 4-10 provides an example of a DPP pipeline for a classification task with a data set that comprises both MVs and categorical features. For better readability, the essential methods are not named in the remainder of this work when providing information about pipelines. A 5-digit identifier is assigned to each of possible 12,160 DPP pipelines. For instance, the pipeline ID for "*MeanImputer+OrdinalEncoder+RemoveCorrelated+SMOTE"* is ID 04946.

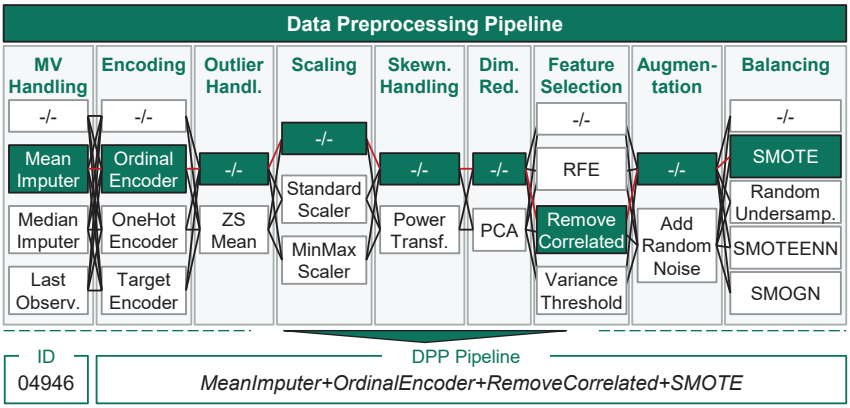


Figure 4-10: Exemplary DPP pipeline including identifier (marked in dark green)

Given the production use cases from Chapter 4.2.1 and suitable DPP pipelines, a benchmarking is conducted to identify best performing DPP pipelines for production use cases.

4.2.3 Benchmarking of DPP Pipelines

In the following the benchmarking setup and its realization is discussed (A). Subsequently, the benchmarking results are presented that reveal best performing pipelines for given use cases (B).

A) Setup and realization

To eventually determine the pool of DPP pipelines that serve as classes of the meta target, the benchmarking aims at finding best performing DPP pipelines for given production use cases. For every of the *1,872 production use cases*, DPP pipelines are

benchmarked. The entire benchmarking procedure is described in Figure 4-11. Within the preparation of the benchmarking, one of 104 data sets is chosen. Dependent on presence of missing values, or categorical features in the data set, as well as the underlying learning task, corresponding DPP pipelines, ML performance metrics and data set characteristics to be acquired during benchmarking are selected (step 2).

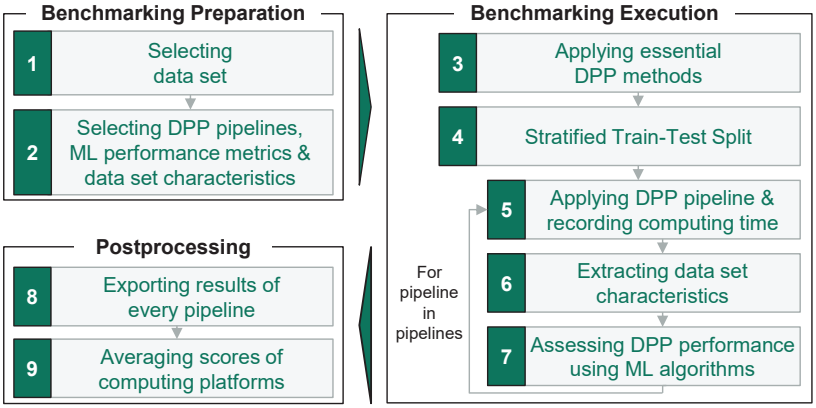


Figure 4-11: Benchmarking procedure

After preparation, benchmarking itself is conducted. According to Figure 4-9, essential methods are applied in advance of the application of subsequent DPP pipelines (step 3). The order of following categories is kept fixed, which lowers the configuration space to keep the number of pipelines in a manageable range. In step 4, the data set is split into a training and test set using 70 % of the data for training the ML algorithm. The 70 %/30 % split accounts for computational costs in training ML algorithms while ensuring representativeness in the test data. Thus, care is taken for a stratified split to ensure class representations in training and test data.

Followingly, steps 5 until 7 show the benchmarking loop. One DPP pipeline is applied at a time. Besides the acquisition of computing times, errors that occur when executing DPP pipelines are acquired (step 5). Data set characteristics are gathered after the application of the DPP pipeline to allow a comparison of data set characteristics prior to and after the application of the corresponding DPP pipeline (step 6). This also supports the interim validation of the correct execution of each pipeline. In the seventh step, the performance of the DPP pipeline is assessed based on the application of 18 ML algorithms. ML algorithms are trained based on default hyperparameter configurations. Repeatable results generated by ML algorithms are thereby ensured through setting the random state. The procedure of step 5 until step 7 is repeated until every

DPP pipeline is called. Hyperparameters of DPP methods are kept fixed during benchmarking. Depending on the learning task and the presence of MVs as well as categorical features, a different number of combinations per data set is benchmarked. If no MVs and categorical features are available in the data set, the number of combinations is equal to 304 for regression and 608 for classification, respectively. The higher number of combinations results from three balancing methods for classification but only one method from regression. While for regression, the maximum number of combinations lies at 2,736, it is 5,472 for classification. In case of either missing values or categorical features are in the data set, number of possible combinations is 912 or 1,824, respectively. The sum of all possible DPP pipelines lies at 12,160. Every pipeline is given an ID. When every pipeline of a given use case is benchmarked, the postprocessing starts.

In the postprocessing phase, data set characteristics and the performance of every pipeline is exported as csv files. For classification tasks, the metrics of recall, precision, accuracy and F1 score are acquired. In case of regression, RMSE, MAE, R^2, and MSE are gathered. Due to the stochastic behavior of DPP and ML, the benchmarking is conducted ten times and subsequently averaged by calculating the mean over the runs (step 10). Annex A.6 further provides an overview about the nine computing platforms being used for benchmarking. The entire code of the benchmarking can be found in Annex B.1.

B) Benchmarking results

For finding the pool of DPP pipelines, which serve as meta target for the meta model, best performing pipelines need to be determined for a given use case. Ideally, the performance of DPP pipelines is identified based on the comparison with performance if no DPP has been executed. However, the direct application of ML algorithms with no DPP leads to an error in 72 out of 104 data sets due to the existence of missing values and categorical features. Therefore, the performance of the essential pipelines functions as a basis to determine the performance of DPP pipelines. As discussed in Chapter 4.2.2, essential DPP additionally can contain imputation and encoding in case data sets exhibit both missing values and encoded features. In the following, the benchmarking results are presented first for classification and afterwards for regression. Performance metrics being used for classification are F1 scores, while for regression, RMSE values normalized by the standard deviation are considered.

In Figure 4-12, F1 scores are shown for 20 representative results regarding classification use cases averaged over ten runs. The whiskers at the top of the box plot as well as outliers above the top whiskers represent the best performing DPP pipeline, while the green markers depict the performance of essential pipelines. The best performing

DPP pipeline for a given use case is the pipeline that leads to the best performing ML model for that use case. The top F1 scores over the 20 representative results are between 0.5 and 1.0.

The four box plots in the beginning (1-4) represent the DPP pipeline scores for four different ML algorithms given the *Accumulator* data set. Regarding the different ML algorithms, different performances are achieved. Based on the data set *GDMS*, *AdaBoost* (9) and *RF* (10) show entirely different ranges of F1 scores underlining the high dependency and mutual influence of ML and DPP. In addition, the box plots of derivatives of the data set *Pump* with *GB* (11, 13, 15, 17) and *SGD* (12, 14, 16, 18) are depicted. Here, the *SGD* performs poorly in comparison with GB for *Pump norm* (11 vs. 12). However, *SGD* outperforms *GB* for *Pump both*. This observation emphasizes that data sets of type *norm* lead to different results.

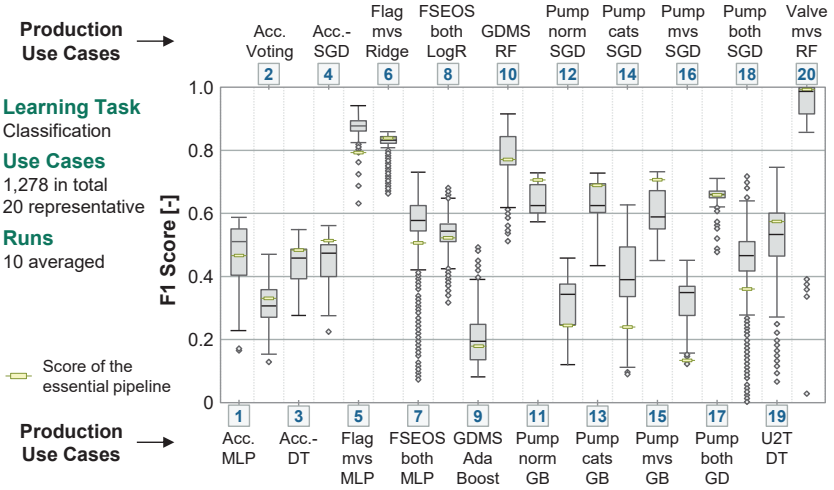


Figure 4-12: Excerpt of benchmarking results for classification use cases

Figure 4-12 shows that best performing pipelines can be found, and for every data set, DPP pipelines outperform essential pipelines. Table 4-6 shows representative, best performing pipelines exemplarily for seven of 1,278 classification use cases. For every data set and ML algorithm, the number of pipelines being benchmarked and the best performing pipeline including its ID are listed. Table 4-6 illustrates the availability of many different best performing pipelines for given use cases.

Table 4-6: Best performing DPP pipelines for a set of use cases

Data Set	ML algo	No. pips	Pip ID	Best performing DPP pipeline	F1 score
Accu-mulator	MLP	608	06576	ZSMean+StandardScaler+PowerTransformer+ PCA+RandomUndersampler	0.59
Flag mvs	MLP	1,824	00582	MedianImputer+StandardScaler+ PowerTransformer+RandomNoise	0.92
GDMS	RF	608	00401	StandardScaler+PCA+RandNoise+SMOTENC	0.91
Pump	GB	608	05601	RemoveCorrelated+RandNoise	0.72
Pump cats	GB	1,824	06242	OrdinalEncoder+ZSMean+MinMaxScaler+ PCA+RandNoise	0.73
Pump mvs	GB	1,824	00136	MeanImputer+ZSMean+StandardScaler+RandNoise	0.71
Pump both	SGD	5,472	06505	MeanImputer+TargetEncoder+ZSMean+ StandardScaler+PCA+SMOTEENN	0.72

ML algo ML algorithm **No. pips** No. of pipelines **Pip ID** Pipeline ID

For instance, the DPP pipeline "*ZSMean+StandardScaler+PowerTransformer+PCA+RandomUndersampler"* achieves a F1 score of 0.59 for *Accumulator* and *MLP*, while the best pipeline for *Pump* and *GB* is "*RemoveCorrelated+RandNoise*" with a F1 score of 0.72. Thereby, pipelines may be very short, e. g., for *Pump* or very long as for *Pump both*. Table 4-6 shows similar F1 scores for artificially created derivatives of *Pump*. However, pipelines again differ to achieve best performances that underlines the meta target tradeoff and justifies the creation of derivatives.

In Figure 4-13, RMSE values normalized by the standard deviation for 15 of 514 use cases averaged over ten runs are shown. The better the DPP pipeline performs, the closer is the pipeline at a NRMSE value of 0.0. For better readability, outlier NRMSE values higher than 2.0 are not considered in Figure 4-13. 2 % of DPP pipeline scores are considered outliers. Data sets such as "Compressor" and "Turbine" reveal high performances, whereas "Milling Table" shows a poor performance close to NRMSE = 1.0.

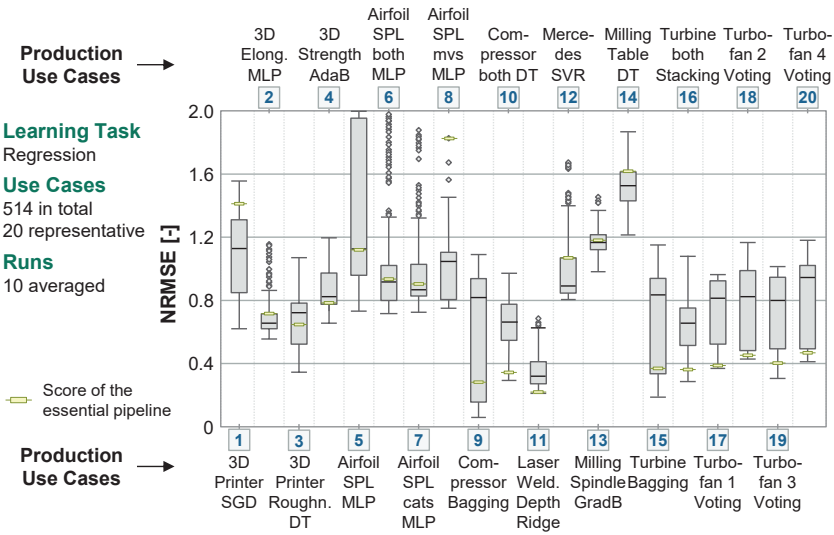


Figure 4-13: Excerpt of benchmarking results for regression data sets

Since Figure 4-12 and Figure 4-13 focus on box plots, one production use case for classification is exemplarily discussed. Figure 4-14 shows the number of DPP pipeline occurrences over the F1 score for the use case *Flag mvs* and *MLP* averaged over ten runs. The essential pipeline is assigned in green with a score of 0.81. The F1 score ranges from 0.62 to 0.92. For the best performing 100 DPP pipelines for this use case, every DPP method is at least called once. One exception represents the encoder methods, since *Flag mvs* do not exhibit categorical columns. A presentation for regression data sets is omitted, since similar conclusions can be drawn from regression data sets. Data set-specific as well as ML algorithm-specific benchmarking results can be found in Annex A.11.

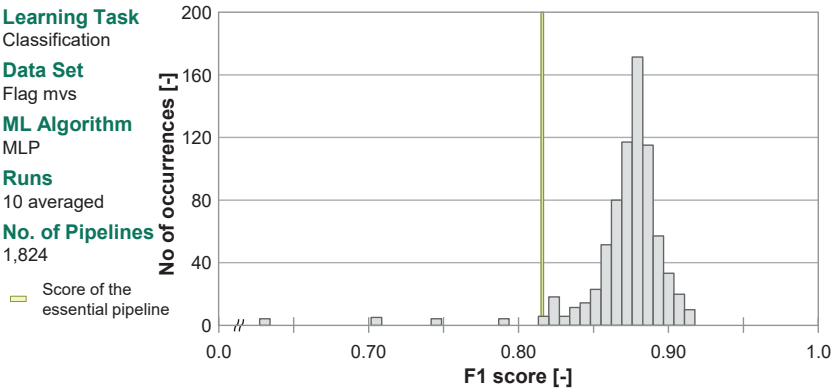


Figure 4-14: Distribution of F1 scores of MLP classifier for *Flag mvs*

Referring to errors, for eight data sets, only ten instead of 18 ML algorithms could be trained due to platform restrictions leading to 64 less use cases. In further 16 cases, *LGBM* was not applicable due to value errors. In total, 1,260 classification, and 514 regression use cases are used for scaled ranking and subsequent pooling ending up in 1,774 production use cases. The mean computing times for executing DPP pipelines lay at 2.6 seconds over all use cases. In addition, only 5 % of all DPP pipelines have a higher execution time than 2.5 minutes, while the maximum computing time for executing one pipeline is 10 minutes. Therefore, computing times will be neglected from here.

In summary, best performing DPP pipelines generally outperform essential pipelines. The range of achieved F1 and NRMSE scores are very high for one use case, while a top performing pipeline for given use cases does not necessarily achieve an overall high score. In addition, many different pipelines perform best over production use cases. Out of a total of 12,160 DPP pipelines, 8,459 pipelines are at least once the top performing pipeline for one of 1,774 production use cases. To address the meta target tradeoff, a scaled ranking is introduced to make the achieved F1 scores and NRMSE values comparable across use cases and to eventually determine overall well performing pipelines.

4.2.4 Scaled Ranking of DPP Pipelines

Due to the high number of best performing DPP pipelines available, overall well performing pipelines are identified to exclude worst performing pipelines and simultaneously reduce the amount of pipelines the two meta models need to choose from. For this reason, a ranking is conducted, which sorts DPP pipelines for each use case in

relation to their performance. The ranks are used to determine well performing pipelines across all use cases.

The DPP pipelines are ranked according to their ML model performance. In practice, ML model performance can highly vary across different data sets ending up in difficult comparison of DPP pipeline performances. For better comparability of achieved performances, a scaling is introduced. The best performing pipeline receives a value of "1.0" and the worst one a value of "0.0". All remaining DPP pipeline IDs are assigned a scaled value between 0 and 1 according to their relative performance. For all 1,774 use cases, in total 12,160 DPP pipelines are ranked by their performance in a scaled manner. Figure 4-15 shows the principle of scaled ranking according to DPP pipeline performances based on the data set *Flag mvs* with a *MLP*.

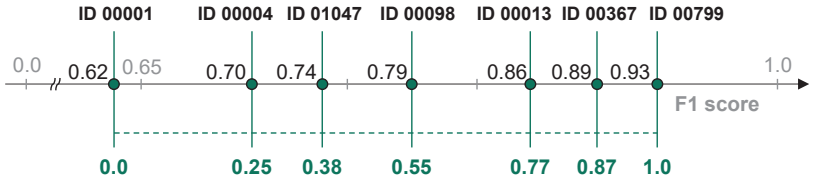


Figure 4-15: Principle of scaled ranking of DPP pipeline performances

DPP pipelines are represented by their IDs, e. g., ID 00001 for F1 = 0.62, or ID 00799 in case of F1 = 0.93. Therefore, ID 00001 receives a scaled rank of "0.0", whereas ID 00799 gets a scaled rank of "1.0". For regression, DPP pipelines with lowest NRMSE values receive the score of "1.0", while the highest NRMSE values are assigned "0.0", since they represent the worst performing pipeline. For every use case, pipelines are assigned a DPP score, in the following called DPP score, according to the principle described in Figure 4-15. To determine overall well performing DPP pipelines, the highest DPP score over all use cases are to be found. Therefore, descriptive statistics are applied to the whole set of DPP pipelines and use cases. For every of 12,160 pipelines, the statistical location measures of min, max, mean, median, 10^{th}, 25^{th}, 75^{th}, 90^{th}, 95^{th} and 99^{th} percentile are calculated over all use cases. A distinction is made between classification and regression data sets. Table 4-7 shows the principle of identifying overall well performing pipelines for classification use cases. Each DPP pipeline is assigned a DPP score for every production use case. At the end, the statistical location measures are calculated based on which the overall well performing pipelines can be derived. In Table 4-7, production use cases follow the naming convention of "*data set – algorithm*". For instance, the first use case is "*Flag mvs – MLP*". The DPP score of DPP pipeline ID 00028 is 0.969. The statistical location measures show

that the pipeline has a minimum DPP score of 0.185, and median of 0.805. Exemplarily, min, mean, median values are shown for the DPP pipelines. To determine overall well performing pipelines, focusing solely on the maximum score is therefore not sufficient. According to the benchmarking results in Chapter 4.2.3, the max score is 1.0 for 8,459 DPP pipelines. In addition, mean and median DPP scores reveal that many DPP pipelines are not overall performant. For instance, the DPP pipeline ID 09374 from Table 4-7 has a maximum score of 1.0, however, mean, and median DPP scores are very low compared to other DPP pipelines.

Table 4-7: DPP scores of DPP pipelines per classification use case

ID	DPP Pipeline	Flag mvs - MLP	Pump mvs - SGD	...	Valve mvs - RF	Min	Mean	Median
00028	MeanImputer+StandardScaler+RandNoise	0.969	0.786		1.0	0.185	0.805	0.864
00204	MeanImputer+StandardScaler+PowerTransformer+RandNoise+SMOTENC	0.950	0.677	...	1.0	0.192	0.766	0.793
⋮		⋮	⋮		⋮	⋮		⋮
04242	MeanImputer+ZMean+StandardScaler+RemoveCorrelated+RandNoise	0.789	0.865		1.0	0.021	0.650	0.738
09374	LastObservedImputer+ZSMean+MinMaxScaler+PCA+VarianceThreshold+RUS	0.478	0.571	...	0.032	0.00	0.467	0.463

RUS Random Undersampler

Production Use Cases | Location Measures

Different statistical location measures can serve as a basis to identify overall well performing pipelines. Figure 4-16 shows the distribution of median DPP scores for the learning task classification. The higher the median score of a pipeline, the better is the overall performance of the DPP pipeline.

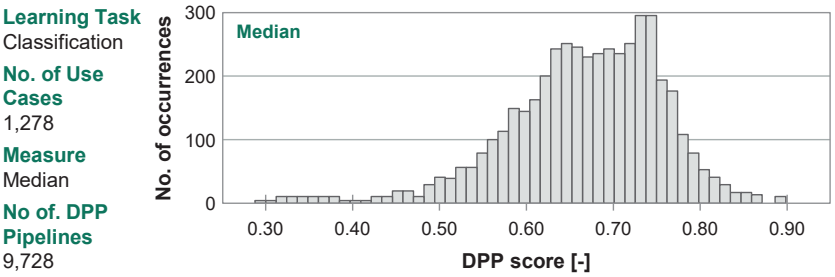


Figure 4-16: Distribution of median DPP scores for classification

For classification, a pipeline is assessed as highly performant, if the median DPP score is higher than 0.75 representing the top 15 % of all DPP pipelines. Poor performing

pipelines only achieve a score of 0.30 according to Figure 4-16. Besides the median DPP scores for every DPP pipeline over all use cases, Figure 4-17 shows the number of occurrences of DPP scores for one well performing (ID 00028) with a median score of 0.864 and one poor performing pipeline (ID 09374) with median score of 0.463 from Table 4-7. The pipelines are based on classification use cases that contain missing values. In both cases, the statistical location measures of 25th, mean, and 75th percentile are mapped. Different performances of pipelines can then be determined by comparing the location measures. For instance, the 25th percentile for ID 09374 is 0.25 and 0.71 for ID 00028. In general, the higher left skewed the distribution of DPP scores, the higher is the overall performance of a DPP pipeline. In addition, the DPP pipelines are highly dependent on the ML algorithm. Therefore, ID 00028 is performant for *MLP* classifiers but less performant in case *AdaBoost* is used. Based on the DPP scores, a pool of overall best performing pipeline can be created, which is detailed in the following subchapter.

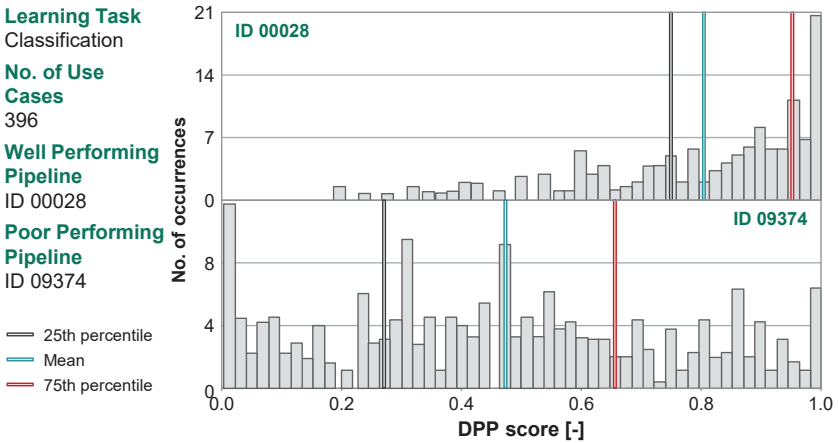


Figure 4-17: Distribution of achieved DPP scores for classification

4.2.5 Pooling of DPP Pipelines

Pooling addresses the meta target tradeoff: On the one hand, a high number of best performing pipelines in the meta target leads to top DPP scores for a given use case but simultaneously many classes and, thus, poor generalization ability. On the other hand, a low number of best performing pipelines results in less classes and better generalization of the meta model. However, top DPP scores for a production use case cannot be achieved. Since a meta model performs generally better in case of more instances per class, and the goal is to find an overall well instead of top performing

pipeline, pooling is applied. In the following, main considerations are outlined before pooling is performed and pooling results are presented.

Main considerations prior to pooling

Since data sets considered by Meta-DPP can be any derivatives with missing values or categorical features as introduced in Chapter 4.2.1, pipelines need to be available that ensure applicability for every data set derivative (1). In addition, pipelines need to be stable so that no errors occur during the application of a recommended pipeline (2).

(1) Availability of DPP pipelines for overall applicability

Production use cases can be distinguished by their derivatives of (*norm*, *cats*, *mvs*, and *both*). The four different cases lead to different DPP categories being applied:

- No encoding and no imputing (*norm*),
- Encoding but no imputing (*cats*),
- No encoding but imputing (*mvs*),
- Both encoding and imputing (*both*).

DPP pipelines are available, which do not exhibit imputing or encoding methods. A pooled pipeline would then not be applicable for new use cases that comprise missing values or categorical columns. DPP pipelines in the pool need therefore be available with imputing and encoding techniques to ensure the applicability for *cats*, *mvs* and *both* data sets.

(2) Error consideration

As shown in Chapter 4.2.3, a minority of pipelines is unstable and therefore should not be considered in the pipeline pool. To ensure the applicability of the recommended DPP pipelines of Meta-DPP, only stable pipelines are recommended. Therefore, a pipeline is a candidate, if the number of errors over 1,872 use cases is less than ten. When applying the constraint, 442 regression and 1,260 classification use cases are finally used for pooling.

Pooling

Based on the main considerations, the pipeline pool can be set up. As proven in Chapter 4.2.3 and 4.2.4, the performance of a DPP pipeline is highly dependent on the ML algorithm, which is considered when developing the pooling method. The pooling method relies on the assumption that a DPP pipeline is overall performant if the pipeline performs well for most ML algorithms. For this reason, the pooling algorithm results as follows:

1. For each ML algorithm over all classification and regression use cases as shown in Table 4-7, calculate DPP scores to obtain 18 DPP score tables
2. For each algorithm-specific DPP score table, calculate statistics over all production use cases, i. e., data sets, and sort by median
3. For each ML algorithm, collect the ten best performing pipelines according to the median over all use cases, i. e., data sets
4. Remove imputers and encoders from all pipelines to focus only on pipelines without imputing and encoding, called base pipelines.
5. Remove duplicated base pipelines if present
6. Integrate remaining base pipelines and count, how often a pipeline occurs over 18 ML algorithms
7. Select *m* pipelines with the highest number of occurrences to be in the pool
8. Search for best combination of imputer and encoder through the overall median performance given the selected base pipeline (see Figure 4-16).

For example, if the pipeline of *StandardScaler* and *SMOTE* represents a pipeline for the pipeline pool, the most performant combination of imputer and encoder, e. g., *MeanImputer* and *TargetEncoder* given the pipeline at hand are added. The resulting pipeline for the pipeline pool is then "*MedianImputer+TargetEncoder+StandardScaler+SMOTE*". Adding DPP methods have no influence in case missing values or categorical features are not present. The computing time for executing these methods is negligible as stated in Chapter 4.2.3.

The selection of *m* pipelines with the highest number of occurrences represents a central hyperparameter of Meta-DPP for handling the meta target tradeoff. The larger *m* and therefore the higher the number of pipelines in the pipeline pool, the more classes are in the meta target. For the development of Meta-DPP, the hyperparameter is chosen to be $m = 4$ for both regression and classification. During the development of Meta-DPP, different hyperparameter values from two to ten were tested in a grid search approach, in which four represents the best tradeoff of having overall performant DPP pipelines in the pool and having a performant meta model, which is capable of recommending DPP pipelines from that pool.

Pooling results

The resulting pipeline pools for classification and regression can be seen in Figure 4-18 sorted by the achieved median DPP scores of the pipelines. Median scores of regression pipelines are higher than classification. Every pipeline from the pipeline pool comprises an imputer and encoder. For seven out of eight pipelines from the pipeline pool, a scaling method is available. Every DPP category is available in the pipeline pool except for *PowerTransformer* and *PCA*. In other respects, the pools differ significantly

from each other. For classification, the combination of imputer and encoder is different, while only *StandardScaler* is used instead of *MinMaxScaler*. *ZSMean* is ones within a top performing pipeline. In case of feature selection, only *VarianceThreshold* is selected, which outperforms *RemoveCorrelated* and *RFE*. The data is augmented through *RandNoise* twice. Every classification pipeline further comprises *SMOTE*, which outperforms *SMOTEENN* and *RandomUndersampler*. The regression pipelines comprise both *MeanImputer* and *MedianImputer* with *OneHotEncoder*. *ZSMean* is twice in top performing pipelines. *StandardScaler* and *MinMaxScaler* is equally distributed. *RFE* outperforms *RemoveCorrelated* and *VarianceThreshold*. *RandNoise* is used twice to augment the data. However, *SMOGN* is not used as balancing method for regression.

ID	Pipeline Pool for Classification	Median
00144	MedianImputer+OneHotEncoder+StandardScaler+VarianceThreshold+RandNoise+SMOTE	0.821
00244	MeanImputer+TargetEncoder+StandardScaler+SMOTE	0.797
00492	MedianImputer+TargetEncoder+ZSMean+StandardScaler+VarianceThreshold+SMOTE	0.771
02390	MeanImputer+OneHotEncoder+RandNoise+SMOTE	0.753

ID	Pipeline Pool for Regression	Median
00005	MeanImputer+OneHotEncoder+StandardScaler+RFE	0.963
00004	MeanImputer+OneHotEncoder+ZSMean+StandardScaler+RFE	0.962
00014	MedianImputer+OneHotEncoder+MinMaxScaler+RFE+RandNoise	0.917
00012	MedianImputer+OneHotEncoder+ZSMean+MinMaxScaler+RFE+RandNoise	0.916

Figure 4-18: Pipeline pools for classification and regression

The pipelines from the pipeline pool can ultimately be mapped to benchmarking results for both classification from Figure 4-12 as well as regression Figure 4-13 in Chapter 4.2.3. The pipelines from the pipeline pool show best performing results over all data sets as shown in Figure 4-19. However, in individual use cases as for instance *GDMS-AdaBoost*, overall performant pipelines show poor performance underlining the complexity of recommending top performing DPP pipelines. All pooling results for classification and regression including use case and ML algorithm-specific distributions can be taken from Annex A.11. Given the two pools of DPP pipelines, a meta target selector chooses between the two pools according to the learning task.

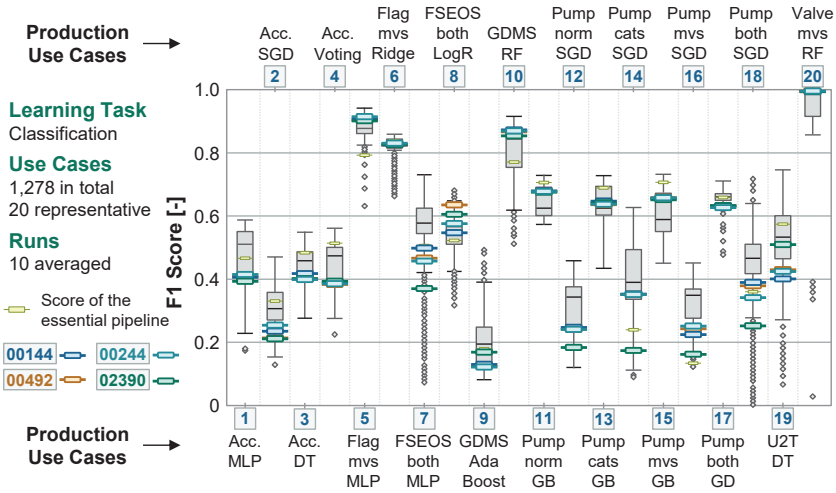
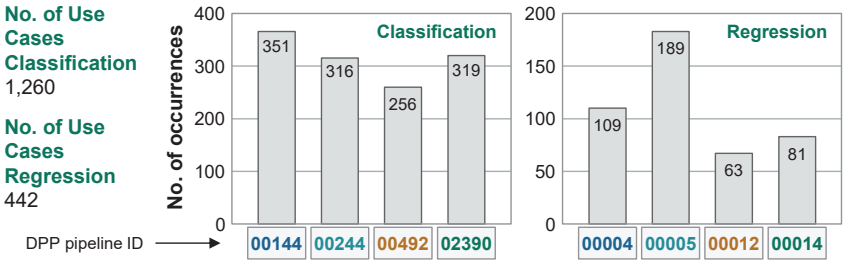


Figure 4-19: Assigned pipelines from pipeline pool for classification

Interim conclusion

Chapter 4.2 focuses on setting-up the meta target selector for Meta-DPP. For this reason, representative production use cases were created, which capture different application areas in production. Production use cases were created based on 104 data sets and 18 ML algorithms. Subsequently, DPP pipelines were configured based on preselected DPP methods. This preselection was carried out based on assessing DPP methods according to eight criteria. The resulting DPP pipelines were benchmarked on 1,872 production use cases. For better comparability of benchmarking results, a scaled ranking was introduced based on which overall performing DPP pipelines were determined. Given the pooling method described in Chapter 4.2.5, four pipelines were preselected for each classification and regression, which serve as pipeline pool. The pipeline pool serves as meta target for the meta model. Since the learning task for the meta model itself represents a multi-class classification task, the resulting class distributions are shown in Figure 4-20. While the learning task classification show only slight imbalances, the regression data set shows an imbalance towards DPP pipeline ID 00005. Ultimately, the meta target selector consists of one rule. If the learning task is regression, the pipeline pool for regression is chosen. Elsewise, the pipeline pool for classification is used. Based on the developed meta target selector and meta targets, the determination of suitable meta features is outlined in Chapter 4.3.



00144 MedianImputer+OneHotEncoder+StandardScaler+VarianceThreshold+RandNoise+SMOTE
00244 MeanImputer+TargetEncoder+StandardScaler+SMOTE
00492 MedianImputer+TargetEncoder+ZSMean+StandardScale+VarianceThreshold+SMOTE
02390 MeanImputer+OneHotEncoder+RandNoise+SMOTE

00004 MeanImputer+OneHotEncoder+ZSMean+StandardScaler+RFE
00005 MeanImputer+OneHotEncoder+StandardScaler+RFE
00012 MedianImputer+OneHotEncoder+ZSMean+MinMaxScaler+RFE+RandNoise
00014 MedianImputer+OneHotEncoder+MinMaxScaler+RFE+RandNoise

Figure 4-20: Resulting class distributions of the meta target

4.3 Developing the Meta Features Database

Given the meta target selector that chooses between two DPP pipeline pools depending on the learning task, the meta features database is developed. Meta features are extracted to systematically obtain useful information from production use cases. Moreover, historical data of ML models as well as DPP pipeline performances on production use cases are used to describe the underlying task. The overall goal of the meta features database lies in enabling the meta model to recommend suitable DPP pipelines for a new production use case. According to Figure 4-21, the meta features database to be developed consists of three dimensions, which are data set, ML algorithm, and DPP pipeline-related.

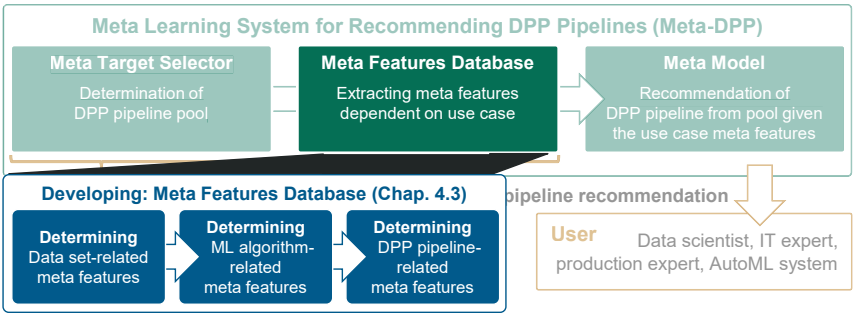


Figure 4-21: Procedure of developing the meta features database

In addition to the structure shown in Figure 4-21, Figure 4-22 provides an overview of the groups within each dimension of the meta features database. Data set-related meta features are directly extracted from the data set.

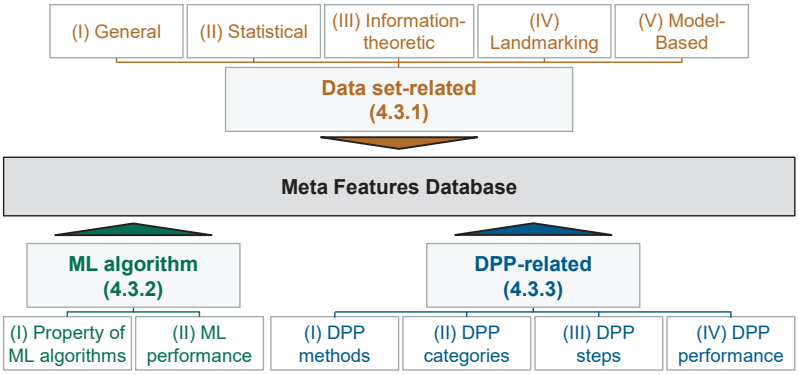


Figure 4-22: Three dimensions including groups of the meta features database

Within three distinct groups, meta features are determined in Chapter 4.3.1, which are based on general, statistical, information-theoretical, landmarking, and model-based information. Chapter 4.3.2 comprises the identification of ML algorithm-related meta features. Starting from general properties of ML algorithms, the benchmarking results from Chapter 4.2.3 are used to further derive tailored meta features that focus on ML model performances. DPP pipeline-related meta features are defined in Chapter 4.3.3, which are based on counting the occurrences of top performing DPP methods in a DPP pipeline. Finally, the DPP pipeline performance of historically top performing pipelines is calculated.

4.3.1 Data Set-Related Meta Features

Data set-related meta features are extracted directly from the production use case at hand according to the three groups shown in Figure 4-22 in the orange color scheme. The extraction is based on the python library *pymfe*, whose documentation also serves as structure in this chapter [ALCO20]. Each group is introduced in the following.

(I) General, statistical, and information-theoretical meta features

General meta features aim at providing a first overview on the data set such as the number of instances and attributes as introduced in Chapter 2.4.2. In addition, categorical and numerical features are used to describe the production use case. For classification, the number of classes and the frequency of each class is acquired, while kurtosis and skewness values of the target variable are gathered in case of regression tasks. 17 general meta features are therefore extracted from the use case. A list of general and further introduced meta features can be taken from Annex A.12.

Based on general meta features, statistical measures can be used to gain knowledge from single features and relationships of multiple features. Referring to single features, statistical location measures such as mean, median, max, or standard deviations are calculated and stored in the meta features database. The meta model requires a constant set of input features for applicability. Since production use cases differ in number of attributes, different numbers of attributes occur per use case. Therefore, statistical location measures of the attributes are further aggregated by calculating the mean and standard deviation of each statistical and information-theoretical meta feature. Statistical measures are further applied to consider the relationship between attributes. Among others, correlation coefficients and covariances are calculated. 51 statistical meta features are acquired for classification and regression.

While statistical measures are mainly focused on continuous features, information-theoretical measures aim at providing information of categorical columns. When considering only one attribute, entropy of attributes and classes is calculated. Joint entropy

and mutual influence are methods that are applied to the data set to identify the dependency between categorical features. In addition, the signal to noise ratio is used to determine the noise of data sets. 13 information-theoretical meta features are used for both classification and regression.

(II) Landmarking

Landmarking aims at providing insights about the task complexity by means of the performance of a learner. Therefore, baseline models in its simplest form are used. Given a data set, e. g., *Flag mvs*, a landmarker is trained, which aims at predicting the label correctly in a supervised learning approach. Landmarking relies on the assumption that the performance of a landmarker provide information about the characteristics of the use case. The error rates of the landmarker's loss function indicate the complexity of the underlying task. Different types of landmarkers have been developed, which do not necessarily aim at achieving low error rates but explore different settings to give insights about the complexity of the task. Therefore, landmarkers as best, random, and worst node, split the data by maximizing and also minimizing the information gain to explore if settings lead to different performances. The higher the difference in error rates is, the less complex is the underlying task. Since seven landmarkers are proposed in literature and available in *pymfe*, they are used to gain insights into the characteristics and complexity of the data sets [ALCO20].

Figure 4-23 shows the number of occurrences (y-axis) of mean error rates (x-axis) for four out of seven landmarkers. The mean error rates on the x-axis result from a 10-fold cross validation during training. In addition, the mean of mean error rates is depicted in red to identify the difference between different landmarker performances.

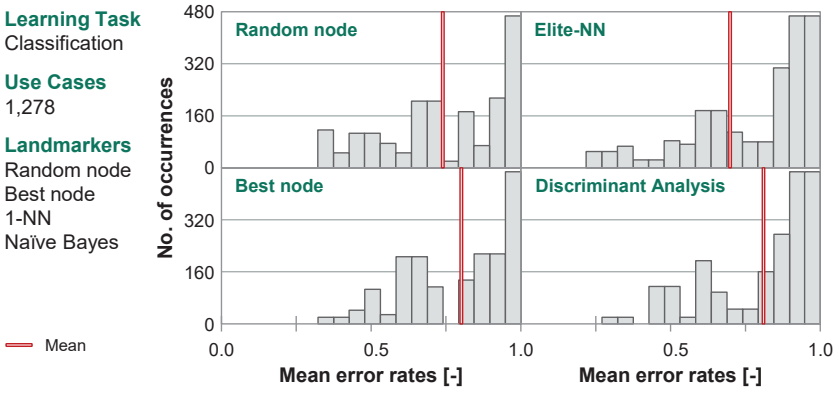


Figure 4-23: Landmarkers used to describe production use cases

The worst landmarker is discriminant analysis, while the best performing is elite-NN. Figure 4-23 illustrates high error rates of the landmarkers and an actual difference of mean performances at 11.8 % underlining the complexity of the meta learning task.

Differences between error rates of landmarkers can also be used to determine the potential success of the meta learning approach. Low performance difference of landmarkers indicate less generalization potential of the final meta model. According to BENSUSAN & GIRAUD-CARRIER, the differences of mean landmarker performances need to be higher than 10 % [BENS00b, p. 3]. Since the difference is slightly beyond 10 %, the meta learning approach can be potentially successful. From each of the seven landmarkers, the mean and standard deviation of error rates are acquired leading to 14 meta features being created through landmarking (see Annex A.12).

(III) Model-based

In addition to landmarking, *decision trees* are trained on the production data set, e. g., *Flag mvs*. The values of the model parameters of the trained *decision tree*, e. g., tree depth, are related to data set characteristics. Model-based meta feature extraction relies on the assumption that model parameters highly depend on data set properties. Different descriptors are used, which are, for instance, *node per attribute*, or *leaves per class*. *Nodes per attribute* calculate the ratio of the number of tree nodes and the number of attributes to indicate how many attributes are used as splitting points. The more splitting points exist, the more complex is the given task. For classification, 24 model-based meta features are extracted. Due to re-occurring errors, no meta features are extracted in case of regression tasks.

When applying Meta-DPP, the values of data set-related meta features are directly extracted from the data set at hand and thereby new for every incoming instance. In case of classification, 117 data set-related meta features are used to describe the task of recommending suitable DPP pipelines. 93 meta features are used for regression.

4.3.2 ML Algorithm-Related Meta Features

In addition to data set-related meta features, information from ML algorithms can be used to describe the task. As for the data set, general properties of ML algorithms can be used to provide initial information. As the pooling of DPP pipelines relies on the performance of ML models, past ML model performances are calculated.

(I) Properties of ML algorithms

Besides the ML algorithm itself, properties of ML algorithms are determined. According to Figure 2-9, ML algorithms can be classified in baseline, ensemble, and *ANN* models.

In addition, *SGD* represents an own algorithm type due to the different learning strategy (see Chapter 2.3.2). Further insights are obtained through the information, whether the algorithm is linear, distance-, kernel- or tree-based, indicating the complexity of the ML algorithm. The robustness against data set properties depicted in Table 2-1 is further used to describe the properties of ML algorithms. 12 meta features are thereby stored in the meta features database.

(II) ML model performance

The ML model performance is used to uncover meaningful information about production use cases. Meta features based on the ML model performance rely on DPP scores achieved during benchmarking over all use cases. The learning task classification is used for explanation in the following. For every of 18 ML algorithms, performances on all classification data sets are considered. Figure 4-24 then shows the approach of extracting algorithm-specific performances for a *MLP classifier*. Given a use case, e. g., *Flag mvs – MLP*, the achieved DPP scores are acquired for every DPP pipeline.

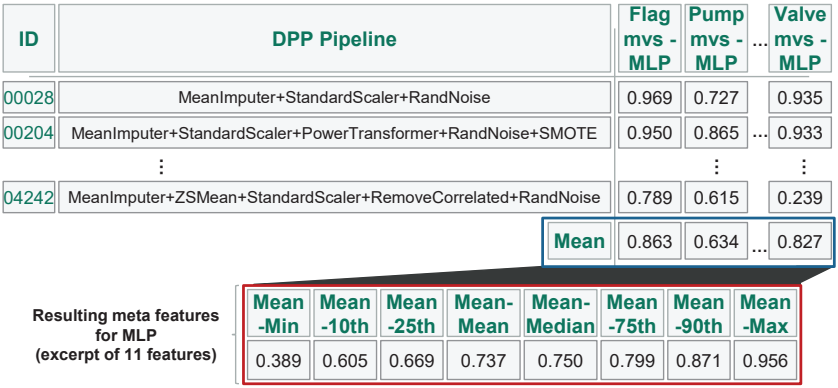


Figure 4-24: Approach of extracting ML model performances from use cases

Subsequently, the mean of achieved DPP scores is calculated over all DPP pipelines. This process is performed for every data set leading to 73 mean values of DPP scores (blue box in Figure 4-24). The mean values are further aggregated to obtain algorithm-specific performance estimates and to ensure that the same set of meta features is acquired independently from the number of pipelines being used. From the mean performance, the statistical location measures are computed that serve as the resulting meta features for every ML algorithm (red box in Figure 4-24). In total, eleven performance measures based on the mean DPP scores of all ML algorithms are used for both regression and classification. Once a meta target is selected, the meta features are retrieved from the database.

4.3.3 DPP Pipeline-Related Meta Features

DPP pipeline-related meta features are those meta features that stem from the performance of DPP pipelines. The creation follows two steps: First, statistical location measures of achieved DPP scores of the best performing DPP pipeline for an ML algorithm are extracted. Second, the number of DPP methods of top three performing pipelines are counted to provide insights about the performance of individual techniques related to a specific ML algorithm. Figure 4-25 illustrates the concept of extracting DPP pipeline-related meta features.

(I) DPP performance

The upper half of Figure 4-25 shows the procedure of extracting eleven statistical location measures of DPP scores of the top performing pipeline (blue boxes). For an ML algorithm, the top performing pipeline is found based on sorting DPP scores according to the median as described in Chapter 4.2.5. In total, 18 top performing DPP pipelines are calculated, while Figure 4-25 shows the top performing pipeline for the *MLP classifier* according to the top median DPP score.

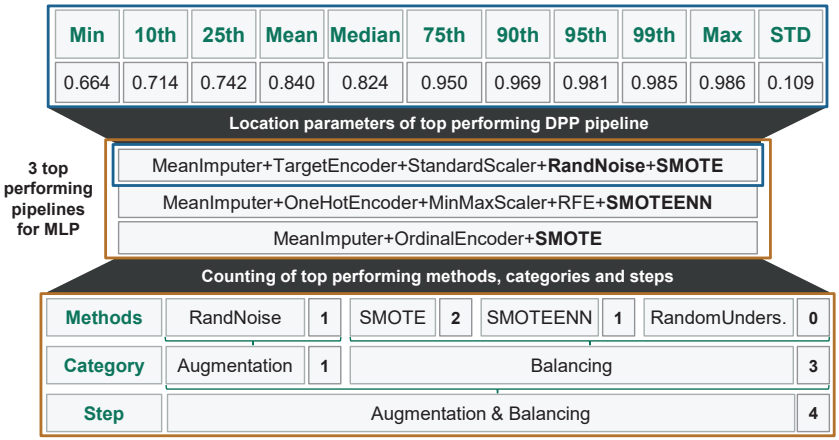


Figure 4-25: Concept of extracting DPP pipeline-related meta features

(II) Counting DPP methods of top performing pipelines

The counting approach is introduced in the lower half of Figure 4-25 (orange). Given three top performing pipelines for one specific ML algorithm, e. g., *MLP*, the occurrences of DPP methods are counted. Based on the taxonomy provided in Chapter 2.4.3, the sum of the counts can be also assigned to DPP categories and DPP steps. In the example of DPP methods of the step augmentation and balancing,

RandNoise and *SMOTEENN* occur once, and *SMOTE* twice in the top three performing pipelines. The *RandomUndersampler* is not prevalent. From method level, the abstraction of category and steps are used to sum the used methods and categories. The counts of 18 DPP methods are used as meta features, while for regression, 16 DPP methods are calculated.

Interim conclusion

The overall goal of the meta features database is to provide suitable information so that a meta model is enabled to recommend suitable DPP pipelines. For that reason, 183 meta features for classification, and 159 for regression have been determined, which were classified in three dimensions: data set-, ML algorithm-, DPP pipeline-related meta features. Ultimately, the resulting meta data set for classification has a shape of 1,260 rows and 183 input features, while regression exhibits 442 instances and 159 input columns. Figure 4-26 shows exemplarily an excerpt of the final meta data set for classification. A focus was made on acquiring many meta features in a first step to obtain information that describe the underlying task successfully. However, too many meta features may lead to the curse of dimensionality as mentioned in Chapter 4.1.1. For that reason, the resulting number of meta features is varied in the design of experiments, when training the meta model in Chapter 4.4.

Index	Algorithm	Inst.	Cor. mean	Entr. mean	Elite_nn .mean	Leaves_ perclass	Type	Mean- mean	Best_ pipe
419	GB	2,205	0.749	3.257	0.795	0.143	tree	0.877	ID 00492
575	KNN	1,109	0.633	2.938	0.305	0.500	instance	0.865	ID 00244
635	LGBM	161	0.170	1.040	0.584	0.500	tree	0.859	ID 00244
820	MLP	1,567	0.283	2.553	0.761	0.500	ANN	0.930	ID 00244
1,001	SGD	1,484	0.283	0.320	0.972	0.500	linear	0.822	ID 00244
⋮	⋮		⋮		⋮		⋮	⋮	⋮
1,260	Voting	9,466	0.255	1.418	0.991	0.500	instance	0.784	ID 00144

Inst. No. of instances Cor. Correlation Entr. Entropy Elite_nn Elite nearest neighbor

Figure 4-26: Excerpt of the final classification data set

While the values of data set-related meta features are directly extracted from the data set at hand and thereby new for every incoming instance, ML algorithm as well as DPP pipeline-related meta features are retrieved from the database based on the outcomes from benchmarking. The latter meta features are updated, when Meta-DPP is retrained. Retraining will be described in more detail in Chapter 4.5. Based on the meta target selector, which chooses between two different pipeline pools according to the

learning task, and the developed meta features database, which serves as input variables, the development of the meta model is outlined in the following sub chapter.

4.4 Developing the Meta Model

Given the meta target and meta features, the meta model can be developed that aims at recommending overall well performing DPP pipelines for ML applications in production. Figure 4-27 shows the development of the meta model within the overall Meta-DPP system.

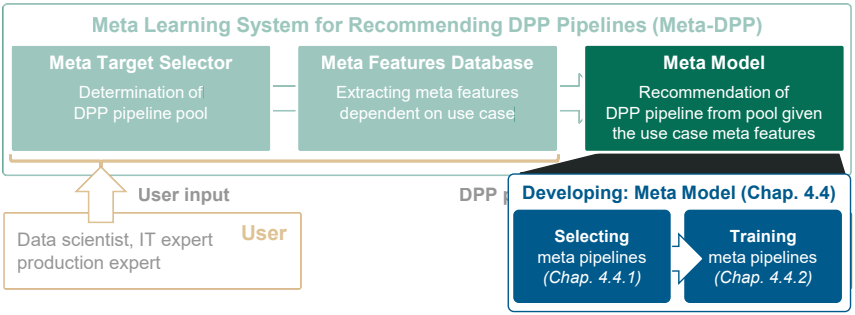


Figure 4-27: Procedure of developing the meta model

Since two different meta targets are available, i. e., DPP pipeline pools for regression and classification, two meta models are set up. As for production use cases, a suitable DPP pipeline and ML algorithm needs to be found in the first step. For that reason, Chapter 4.4.1 discusses the selection of a suitable meta pipeline. The training of the meta pipelines is presented in Chapter 4.4.2.

4.4.1 Selecting Meta Pipelines

The selection of meta pipelines is based on the findings of the benchmarking and addresses the requirement of complexity reduction. Since the meta data sets contain categorical features in nominal scale and low cardinality, e. g., the type of ML algorithm, a *OneHotEncoder* is chosen over *OrdinalEncoder* and *TargetEncoder*. For the selection of the meta model being trained, explainable ML algorithms that provide ranked recommendations are chosen. First, baseline models of *DT* and *logistic regression* are selected. Secondly, commonly used meta models are trained, which are *K-NN*, *boosting* algorithms such as *LGBM* and *GB* as well as *extra trees classifier*. In total, six meta pipelines for both regression and classification consisting of a *OneHotEnocder* and six ML algorithms are chosen.

4.4.2 Training Meta Pipelines

First the meta pipelines are implemented for training (I). Secondly, the training results are analyzed (II).

(I) Implementing meta pipelines

After applying the *OneHotEncoder* with default hyperparameter settings, the ML algorithms are trained using a k-fold cross validation. In case of classification, a seven fold cross validation is selected, while for regression, four folds are used. Per fold, it is ensured that seven to eight data sets are considered. The allocation of data sets to each fold can be taken from Annex A.13. Care is taken that derivatives of data sets are not available in both train and validation folds, i. e., if a data set is chosen for validation that contains derivatives of *mvs*, *cats*, and *both*, all data sets are allocated to the validation fold.

After applying the *OneHotEncoder* for the classification pool, 2,048 rows and 201 columns exist, while for the regression pool, 620 instances and 174 attributes are available. These data sets are used to train the six ML algorithms. During training, feature importance is acquired to identify the most important features based on which the ML model recommends the DPP pipeline. Based on the important features, ML models are trained with different number of input features, which range between 20, 30, and all features. To identify the importance of engineered meta features of pipeline-related meta features, the ML algorithms are trained in different configurations as shown in Figure 4-28. Since meta features regarding the data set and ML algorithm are deemed important in existing approaches (see Chapter 3.2) and to keep the number of experiments in a manageable range, only pipeline-related meta features and all meta features are considered in the design of experiments.

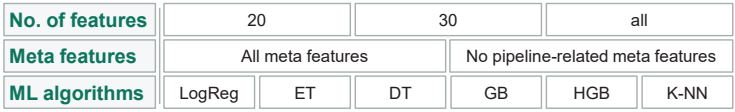


Figure 4-28: Design of experiments for finding the best performing meta model

For every ML algorithm, probability functions are called to receive the probability of how certain an ML algorithm is in recommending the DPP pipeline. In addition, the probability serves as the final ranking of proposed pipelines.

(II) Analyzing the training results of meta pipelines

In the following, the training results are analyzed, which comprises the analysis of feature importances to give an indication on how the final meta model performs on unseen test data. Three different sets of meta features are used for training the meta model. The feature importance, which ranges between 0 and 1, is low over all sets of meta features with a maximum value of 0.08. First, the low value indicates that features are

deemed equally important by the model. Secondly, it underlines the high task complexity since the features provide little information on which DPP pipeline to choose. Thus, the feature importance results confirm the high task complexity shown during the landmarking results.

When considering all meta features, the 30 most important features are distributed over all dimensions, i. e., data set-, algorithm-, and pipeline-related. In case no pipeline-related meta features are used, the importance of algorithm-related meta features significantly increases. For instance, the *extra trees classifier* relies on 28 algorithm-related meta features when predicting the DPP pipeline pool for classification. All ML models show almost no feature importances indicating to not learn from the features. In particular, *logistic regression* show negative feature importances after 13 important meta features indicating underfitting during training. An overview of feature importances is provided in Annex A.13. In conclusion, a tendency can be derived that tree based algorithms such as *decision tree* and *extra trees* outperform other ML algorithms. However, since the meta pipeline performance can only be identified based on unseen test data, the final choice of the most performant meta model can only be made after the testing on new data sets. Therefore, the developed meta pipelines are applied on test data, which will be discussed in Chapter 5.

Concept for retraining Meta-DPP

Since the developed meta pipelines and the core components of Meta-DPP are designed to be retrainable and updatable, a retraining concept is introduced in Figure 4-29 that aims at ensuring the application of Meta-DPP on the long term. Given a new production use case, Meta-DPP is applied to get the recommended pipeline y_{pred} based on the pipeline pool $pool_{pred}$. Simultaneously, the benchmarking is run for every new production use case. Even though the user may provide a use case with only one ML algorithm, the benchmarking is always conducted with 18 ML algorithms. Best performing pipelines are then calculated based on the DPP score. As described in Chapter 4.3, the meta features database contain meta features that are derived from benchmarking and the scaled ranking. Thus, meta features are recalculated in this phase. The pooling of pipelines according to the method provided in Chapter 4.2.5 is eventually performed. Based on the DPP score for the given use cases and the recalculated pool, y_{act} and $pool_{act}$ are available.

The retraining concept consists of two ways. The first approach (I) is followed, if the $pool_{act}$ contains the same pipelines after recalculation as $pool_{pred}$. Then, the meta model is retrained in a supervised learning approach based on the updated meta features database. In the second approach, the pipeline pool $pool_{act}$ has changed. This can come from two different reasons. Either the hyperparameter of the pipeline pool is

adjusted, or the updated pipeline pool contains different pipelines. In the latter case, the meta target changes, which requires the update of all Meta-DPP components and retraining of the meta model. Instead of updating model parameters of the meta model, a benchmarking of selected meta pipelines is applied to identify the top performing ML pipeline, which then serves as new meta model. A new meta model may also concentrate on a different sub set of meta features. Besides the retraining mechanism, the benchmarking can be updated with new DPP methods and ML algorithms. In addition, new meta features can be calculated or determined by bringing in domain knowledge as described in Chapter 2.5.

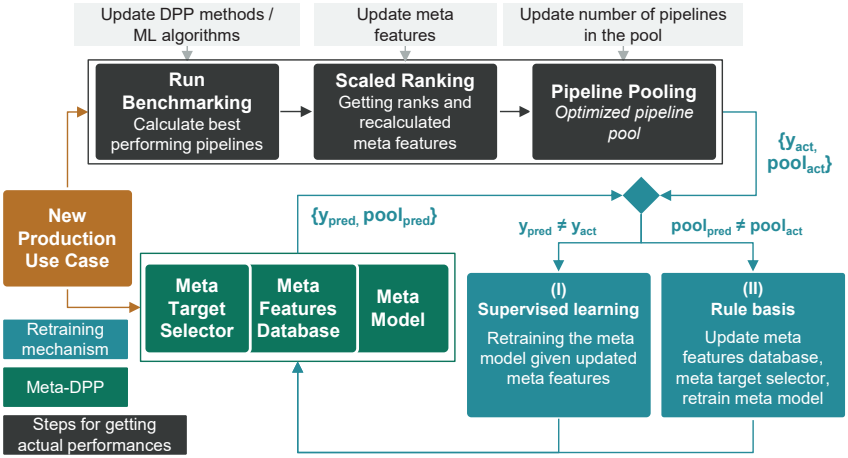


Figure 4-29: Concept for retraining Meta-DPP

Interim conclusion

After selecting the top performing meta pipeline, the meta pipelines for regression and classification are implemented. Ultimately, all core components of meta target selector, meta features database, and meta model are developed, which can be implemented. The implementation is introduced in Chapter 4.5. The developed Meta-DPP is then verified and validated in Chapter 5.

4.5 Implementing Meta-DPP

The developed Meta-DPP consisting of meta target selector, meta features database and meta model is eventually implemented to be applicable by a user. For the implementation, python 3.9 and several corresponding libraries are used. To comply with the requirements of having a high usability, a graphical user interface (GUI) is built based on the python package *streamlit* [STRE22]. The entire source code can be taken from Annex B.2. The user provides the necessary information of the production use case via the GUI that contain the data set, learning task, ML algorithm and information about the necessity of having only explainable DPP methods. The selection is realized through a drop down menu. The GUI with the drop down menu can be taken from Annex A.14.

The backend consists of two classes: a *meta instance constructor* class and a *meta model* class. The *meta instance constructor* takes the input from the user to create an instance that fits the dimensions and specifications of the data used to apply the *meta model*. To transform the data set into a functioning meta instance, the method *construct data set* is used. Depending on the learning task, data set-, ML algorithm-, and DPP pipeline-related meta features are generated. For the extraction of data set-related meta features, the extraction library *pymfe* is used [ALCO20]. Error-prone meta features described in Chapter 4.3 are subsequently removed. In case missing values are present in the data point, the values are imputed based on the representative production use cases from Chapter 4.2.1. Depending on the selection of the meta pipeline, more features can be removed from the instance by considering *n* features with the highest feature importance. Ultimately, the meta instance is one hot encoded.

The *meta model* class is the main object that the GUI interacts with. This class contains the class attribute *meta instance constructor*. Furthermore, the *meta model* class hold the method *make data set,* which calls the method *construct data set* from above to receive the newly created meta instance. This new data point is then used as a parameter for the *make recommendation* method. Here, the trained meta model is loaded and used, which aims at recommending the best performing pipeline from the pipeline pool. The user finally receives the main outcomes of Meta-DPP: The pipeline pool including the recommendation probabilities from which a user takes the order of pipeline performances.

4.6 Interim Conclusion

Chapter 4 comprised the development of the meta learning based Meta-DPP for recommending DPP pipeline for ML applications in production. The general design of Meta-DPP was provided in Chapter 4.1. Based on fundamental design considerations, the core components of Meta-DPP and its interactions were discussed. The core components of Meta-DPP are represented by the meta target selector, meta features database and meta model, which provides the answer for SRQ 3.

SRQ 3 Which core components can form a decision support system for recommending DPP pipelines in production (Chapter 4.1)?

In Chapter 4.2, DPP methods were selected by assessing DPP methods based on eight decision criteria. Each DPP method was assigned a mean score, while DPP methods of a DPP category are chosen if they achieve the highest assessment score. In case too many homogeneous methods were present, a benchmarking at small scale was carried out to reduce the configuration space in subsequent benchmarking. Once selected, DPP methods were configured into DPP pipelines while considering the order of DPP steps and conditionalities. Ultimately, 21 DPP methods were chosen and configured into DPP pipelines. Dependent on the learning task and the existence of missing values as well as categorical features, the number of DPP pipelines varies from 304 up to 5,472. The entire list of DPP methods and how these methods are combined into DPP pipelines can be taken from Chapter 4.2.2, which eventually answers SRQ 4.

SRQ 4 Which DPP methods are most suitable for preprocessing production data and how can DPP methods suitably be configured into DPP pipelines (Chapter 4.2)?

A focus of Chapter 4 was put on developing Meta-DPP's core components including meta target selector, meta features database and meta model. The meta target selector development was performed in five steps. First, 1,872 production use cases were created based on the combination of data sets and ML algorithms. Subsequently, DPP pipelines were configured. For every production use case, DPP pipelines were benchmarked. The performance of DPP pipelines were determined based on the performance of ML algorithms. To identify overall best performing pipelines, a scaled ranking was applied. To address the meta target tradeoff, a pooling was performed that takes into account the high dependence between ML algorithm and DPP pipelines. Finally the meta target selector chooses between two pipeline pools of either regression or classification.

Subsequently, the meta features database was set up, which consists of data set-, ML algorithm-, and DPP pipeline-related meta features. While data set-related meta features are calculated based on the data set at hand, historical performances of ML algorithms and DPP pipelines were calculated from the benchmarking and scaled ranking results. Finally, the meta model was developed in two steps. First, suitable ML pipelines were identified and secondly trained. The ML pipelines for the two pipeline pools were finally implemented. Therefore, SRQ 5 can be answered.

SRQ 5 How can the components of the decision support system be set up? (Chapter 4.2, Chapter 4.2.4, Chapter 4.4)

Based on the implemented Meta-DPP as described in Chapter 4.5, Meta-DPP can be verified and validated. Chapter 5 therefore comprises the critical assessment of Meta-DPP.

5 Verification and Validation

Given the development and implementation, Meta-DPP is verified and validated. Verification can be defined as the “[…] formal proof of the correctness of the simulation model” [VDI-18]. According to DAVIS, it is checked if the model implementation correctly reproduces the “[…] developer’s conceptual descriptions and specifications” [DAVI92]. In addition, validation investigates “[…] if the model reproduces the behavior of the real system accurately enough and without error" [VDI-18]. BALCI points out that proving the correctness of a simulation model, here Meta-DPP, cannot be formally completed due to its high complexity [BALC98].

For that reason, multiple techniques have been developed by BALCI and RABE ET AL. that support the verification and validation of Meta-DPP. These techniques are applied during and after development of a software and are sorted dependent on the degree of subjectivity. Each technique supports to increase the confidence in Meta-DPP, however, no general procedure exists for the choice of suitable techniques. Annex A.14 provides an overview of techniques for verification and validation, which are classified in informal, static, dynamic, and formal according to BALCI. Informal techniques draw on human reasoning and subjectivity, whereas static techniques assess the accuracy on the basis of model design and source code characteristics. In terms of dynamic techniques, the model is evaluated based on the behavior when being executed. A mathematical proof of model’s correctness is performed in formal techniques. [BALC98], [RABE08]

Informal techniques have been used during the development of Meta-DPP. Desk checking, or self-inspection, and face validation were performed in every step when developing the core components and realizing the final implementation (see Chapter 4). Within *static techniques*, data inconsistencies were checked and fault analyses carried out especially in case of benchmarking 1,872 production use cases in Chapter 4.2. This process is also coherent with the research approach and statements of MAYRING outlined in Chapter 1.3, who characterizes the development as continuous information acquisition, constant questioning, and practical actions. [MAYR02, pp. 51-54] Debugging and execution testing relate to *dynamic techniques* and were performed in Chapter 4.2-4.5.

Given the techniques, Meta-DPP is further verified and validated based on a case study in this chapter, which relies on multiple production use cases (Chapter 5.1 and 5.2). For further verification and validation, different forms of execution and special input testing proposed by BALCI are performed to assess the correctness of Meta-DPP (see Chapter 5.3) [BALC98].

5.1 Validation through Case Study

Meta-DPP is verified and validated through a case study to determine the performance on production use cases. The general procedure and setup of the verification and validation is discussed in Chapter 5.1.1. Subsequently, the five scenarios considered in this case study are described in Chapter 5.1.2-5.1.6. Thereby, multiple verification and validation techniques are covered according to BALCI, which are acceptance testing, functional testing, performance testing, and product testing [BALC98].

5.1.1 General Procedure and Setup

Figure 5-1 summarizes the procedure of validating Meta-DPP. Since it is essential that the meta target contains actually high performing pipelines, the components that create the meta target need to be validated, namely benchmarking of DPP pipelines (1) and pipeline pooling (2). Lastly, the recommendations of Meta-DPP (3) are validated.

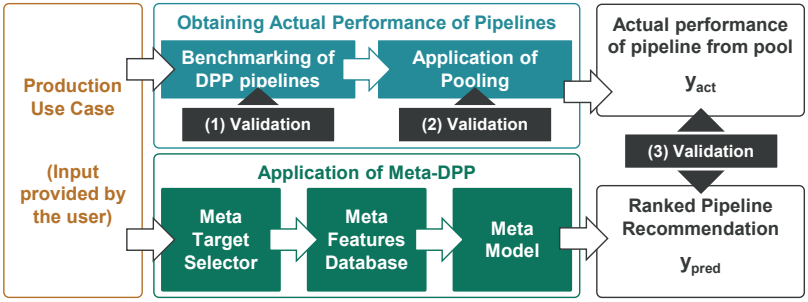


Figure 5-1: Procedure for validating the performance of Meta-DPP

(1) Validation of DPP pipeline benchmarking

The best performing DPP pipeline from the benchmarking should outperform essential pipelines. Essential pipelines consist of essential methods described in Chapter 4.2.2. In case of missing values and categorical columns, imputation and encoding are considered as essential methods. Since different imputation and encoding techniques can be selected, the average over all essential methods is calculated and compared with top performing pipelines. The benchmarking of DPP pipelines is conducted up to seven times for every production use case given computation restrictions, whose results are averaged after benchmarking to address the randomness of DPP pipeline performances. Benchmarking conditions are kept fixed as stated in Chapter 4.2.3. In addition, it is verified whether the pipelines from the pool and essential pipelines do not raise errors during benchmarking.

(2) Validation of DPP pipeline pooling

The pipeline pool is a preselection of four DPP pipelines based on defined criteria from Chapter 4.2.5. For validation, the pooling should contain at least one pipeline, whose achieved mean DPP score outperforms essential pipelines. Pooling is performed as described in Chapter 4.2.5.

(3) Validation of Meta-DPP recommendations

For every production use case, which is provided as input from the user, two strands of *obtaining actual performance of pipelines* and *application of Meta-DPP* are followed. This is represented by the lower and upper walkthrough in Figure 5-1. Recommendations y_{pred} are provided by Meta-DPP through the execution of the meta target selector, meta features database and meta model (lower walkthrough). In parallel, the actual recommendations are calculated y_{act} (upper walkthrough). Actual recommendations are gained through benchmarking, scaled ranking that is not depicted for simplicity reasons, and pooling. Note that the computing time for the upper walkthrough takes up to three days, while the execution of Meta-DPP takes less than one minute on average ($t_{upper} >> t_{lower}$). The ranked pipeline recommendation can be compared with the actual rank of pipelines from the pipeline pool. Thereby, the meta model's suggestion should outperform essential pipelines.

Production use cases

For validating Meta-DPP, production use cases are defined based on four recent and past projects conducted at the Fraunhofer IPT as well as on one publicly available data set. The set of validation use cases are selected based on the following criteria:

- Covering a high variety of production domains (e. g., different production processes)
- High variety of ML applications in production (see Chapter 2.1)
- Representative use case, i. e., high number of pilot demonstrations realized in the production environment
- Existence of classification and regression tasks
- Existence of data set derivatives (*norm*, *cats*, *mvs*, *both*)

When considering the criteria, five scenarios are selected, which contain five different production domains as shown in Figure 5-2. Three ML applications are related to product quality prediction due to the high amount of existing use cases in production [WEIC19]. Regression and classification as well as all derivatives of data sets (*norm, cats, mvs, both*) are included. Figure 5-2 shows the overview of five scenarios mapped to the ML use cases in production as introduced in Chapter 2.1. One scenario thereby consists of one or multiple data sets, while for every data set, 18 ML algorithms are

trained. If one scenario contains one data set, 18 production use cases are used to validate Meta-DPP.

Blisk milling represents the first scenario, in which the tool path deviation is predicted based on planning data. One data set is considered for this case. The product quality from space launcher production (five data sets), wind turbine manufacturing (two data sets), and tool and die making (one data set) are predicted resulting in 8 data sets for product quality prediction.

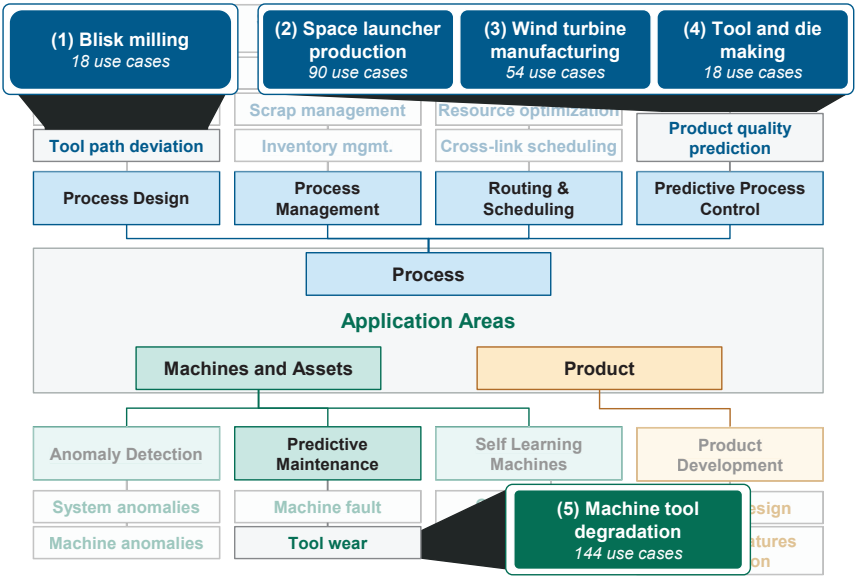


Figure 5-2: Scenarios considered for validating Meta-DPP

Within predictive maintenance, blade degradation of a packaging machine is predicted leading to eight data sets. Given the five scenarios, 18 data sets are used for validation. Different data sets per scenario arise if different product types, further processes, or adjusted quality specifications are available. In total, 324 production use cases are considered. Each scenario including the data sets is described in detail in Chapter 5.1.2-5.1.6.

5.1.2 Scenario 1: Blisk Milling

The first scenario focuses on the production of jet engine components. In simplified form, a jet engine consists of a fan, a compressor, a combustion chamber, a turbine, and a nozzle. Depending on the concept, two to three shafts rotate on which the com-

pressors and turbines are mounted. Rotating compressor components ensure compression of the inflowing air. A compressor consists of several stages, which has a rotating and static unit. The rotating stages have to withstand high stresses due to high rotational speeds and forces. This requires sophisticated compressor design to be manufactured to meet the requirements of the aviation industry. Whereas in the traditional fir-tree design, blades are mounted onto a disk, a blade integrated disk (Blisk) is a single component. Advantages are the higher load capacity and resistance. [SPIT03]

Blisks are predominantly machined through 5-axis milling. To avoid vibrations, Blisks are milled in a block-strategy. For this purpose, every blade of the Blisk is divided into different blocks. Each block is milled in three steps. Roughing removes most of the material, while pre-finishing generates the contour with a predefined stock allowance. This is followed by finishing, which is responsible for the final surface roughness and profile deviation. [MINO15] Among others, the profile deviations depend on the precision of the tool path, which is influenced by the tool and workpiece deflection during machining. To reduce the profile deviations, the tool path deviation can be predicted to adjust parameters such as spindle speed within the machining process. The process adjustment then leads to fewer deviations. Planning data from CAM programming is used to predict toolpath deviations. [KLOC15]

The prediction of the tool path deviation constitutes a regression task. Due to its high influence on the profile deviations, the focus lies on the finishing operation. A block of a corresponding blade is randomly selected. The data set stems from a project and has 13,873 instances and 75 columns, with the last column being the target variable. The standard deviation of the target variable is calculated to standardize the RMSE values for better comparability of ML model performance. There are neither missing values nor categorical columns. During benchmarking, 18 ML algorithms are trained, resulting in 18 use cases. For every use case, 304 DPP pipelines were benchmarked.

Since the data sets of planning and processing are in two different coordinate systems, coordinate transformations were performed. Furthermore, the frequency of the data points during the transformation of the data has been adjusted in such a way that as a result an equal number of data points was available for planning and processing. A summary of the Blisk milling scenario can be found in Figure 5-3.

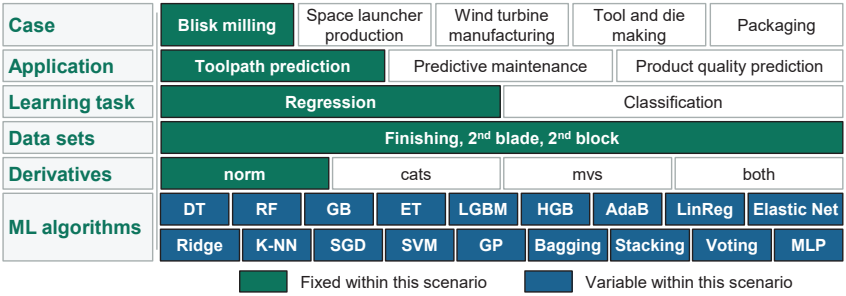


Figure 5-3: Creation of 18 use cases from Blisk milling

5.1.3 Scenario 2: Space Launcher Production

To ensure independent access to space, new launcher generations need to stay competitive in the global market. One promising approach to ensure global competitiveness is the application of AI in production to increase efficiency, product quality and reduce production costs. The Fraunhofer IPT in Aachen, Germany, optimizes the manufacturing of the upper stage, called *Upper Liquid Propulsion Module*, of the Ariane 6 in close collaboration with ArianeGroup in the BMWK-funded project "*SPOK*" between 2020 and 2022 [SPOK22].

The Ariane 6 is a cryogenic launcher that consists of two to four solid boosters at the bottom of the launcher, a cryogenic main stage, a cryogenic upper stage, and a payload stage. The upper stage contains a liquid hydrogen (LH2) and liquid oxygen (LOX) tank, which are connected through an inter tank structure. The production of the upper stage consists of manufacturing different tank types and structures, which is characterized by long process chains and long-lasting processes. Processes are carried out partially automated in different stations. Due to big dimensions, transport routes between individual stations are time consuming and costly. In SPOK, the aim is to predict the surface quality of the tanks and inter tank structures at an early stage in the process chain to improve overall production efficiency. If problems are detected at an early stage, reworking can directly be carried out at the corresponding station. In addition, process parameters can be adjusted during or prior to the processing to increase product quality. [IPT21]

During tank production, the surfaces of different tank types and inter tank structures are treated. Subsequently, an insulation is applied to the treated surface to protect the tank from too high temperature gradients during operation. In the final stage, the quality is inspected by measuring the thickness of the insulation layer. Since insulation thickness represents a key quality characteristic, it is used as continuous target variable. In

conclusion, the learning task of this ML use case is regression. Process parameters during the machining of the surface treatment and application of insulation layer serve as input parameters for 18 ML algorithms. The simplified process chain including exemplary input features and target variable can be taken from Figure 5-4. Due to different tank models and process sequences, nine different products are created and considered as production data sets, which are namely:

- Liquid hydrogen (LH2) – model 1A (LH2-M1A)
- Liquid hydrogen (LH2) – model 1B (LH2-M1B)
- Liquid hydrogen (LH2) – model 2 (LH2-M2)
- Lower inter tank structure (LITS) – model 3 (LITS-M3)
- Upper inter tank structure (UITS)– model 1 (UITS-M1)

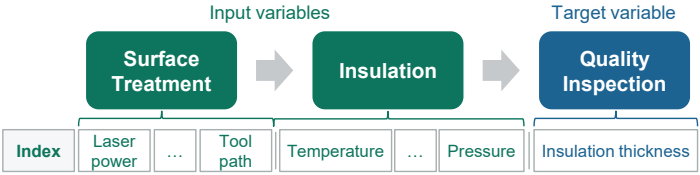


Figure 5-4: Simplified process chain and corresponding variables

Nine data sets and 18 ML algorithms are used for validating Meta-DPP leading to 90 use cases. The number of data set attributes range between eleven and 15. Every feature of the nine data sets is numerical, while no missing values are present. Therefore, neither encoding nor imputation is necessary for the use cases resulting in 304 pipelines to be configured within this scenario according to Chapter 4.2.1. In addition, the number of instances range between 2,978 and 4,996. Due to the normalization of RMSE values in regression tasks for better comparability of performances over use cases, the standard deviation values of the insulation thickness are acquired. Figure 5-5 provides a summary of the space launcher production scenario.

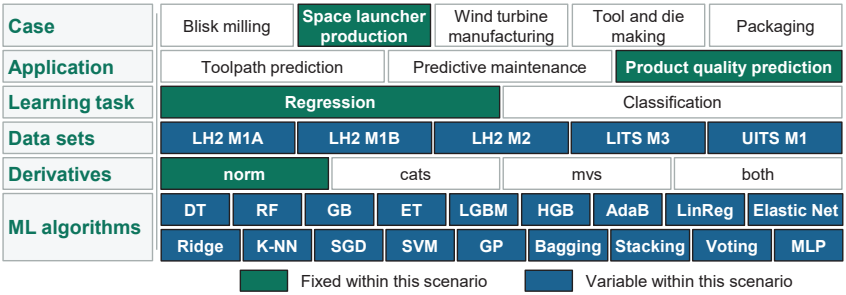


Figure 5-5: Use case creation based on space launcher production

5.1.4 Scenario 3: Wind Turbine Manufacturing

Wind turbines play a key role in the energy change strategy of the German government. A wind turbine consists of a rotor, nacelle, tower and generator that converts mechanical energy into electrical energy. Like airplane wings, usually three rotor blades use the flow of the wind to generate lift, which causes the rotor to rotate. A nacelle is mounted on the tower and can be rotated into wind direction. The nacelle houses the components including the generator to eventually convert mechanical into electrical energy. Rotor blades are made of glass fiber reinforced composite material, which is lightweight and has high fatigue strength. Stabilizers inside the rotors are used to increase their strength and to withstand extreme weather situations. [WIND22]

A project focused on the manufacturing of stabilizers. The stabilizers are manufactured in a three-stage process. In the first step, recycled plastics are extruded in a continuous process, which intermediate good is further glued in two subsequent processes. At the end of the process chain, the product quality is assessed. Figure 5-6 shows the process chain and corresponding data sources.

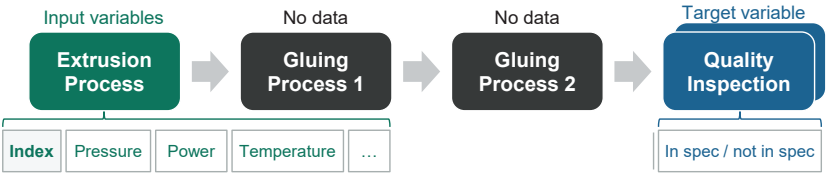


Figure 5-6: Process chain including data sources of rotor stabilizer production

By means of seven characteristics, the product is classified as being within or out of specification. Two different tolerance settings are used to assess the product quality. Data is acquired only in the extrusion process, since the gluing processes are manually performed. However, the extrusion process is deemed to have the highest impact on the product quality. The prediction of the product quality represents a two-class classification. Due to the different tolerance settings and arbitrarily set thresholds, two different target variables are available leading to two data sets. The first data set has a class representation of 32:87, while the second exhibits an imbalance of 53:66. Both data sets contain 238 rows and 37 columns. Encoding is required due to two categorical columns in the data set. Besides the very low number of instances, the low ratio of number of data points to the number of features represents a quality issue.

Because of the continuous process, the link between quality and process data cannot be realized through location of data points on the product as performed in Blisk milling. The traceability of the product is realized by using time stamps of the machine and by

marking the product subsequent to the extrusion process. Figure 5-7 summarizes the creation of 36 production use cases from wind turbine manufacturing.

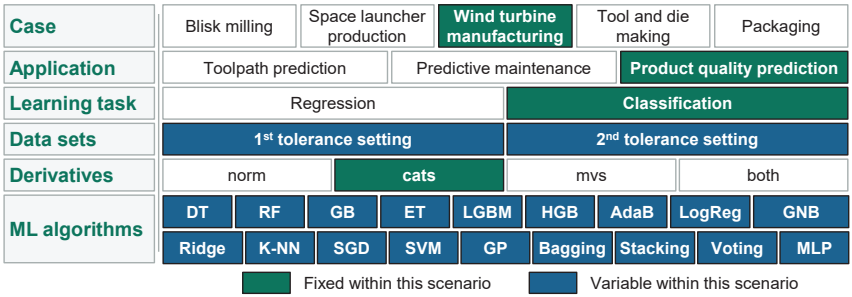


Figure 5-7: Creation of 36 production use cases from wind turbine manufacturing

5.1.5 Scenario 4: Tool and Die Making

Another project was carried out in the field of tool and die making focusing on the production of machine spindles. A spindle is a rotating axis within a machine tool. Spindles are used to facilitate the standard use of tool holders to automatically change tools during machining. Spindles represent the interface between tool and motor of the machine. Therefore, spindles are operated under high loads, which requires spindles to be of high quality in terms of surface and deviations. The spindle itself contains several parts, which need to be manufactured and assembled. Figure 5-8 illustrates the process chain of the spindle production including its data sources.

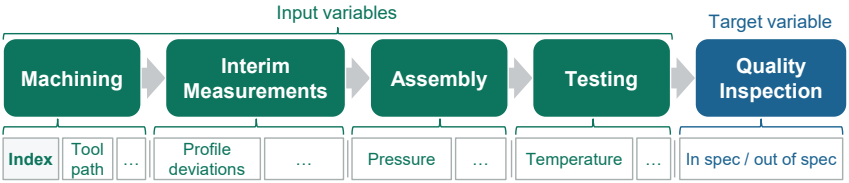


Figure 5-8: Process chain and data sources for predicting the quality of spindles

The spindle is produced in four steps, consisting of machining, interim measurements of incoming parts, assembly and testing. The target variable is the product quality, whose information is provided by the customer in the incoming inspection or during the warranty period. To reduce the requests and complaints by the customers, the goal is to increase the product quality by correlating and predicting quality parameters with input variables. In every process step, data is acquired, while the vast majority is acquired during the assembly step. The final data set contains 233 rows and 73 columns.

Every row is represented by a spindle, while the columns contain data as shown in Figure 5-8. Besides the low amount of rows, 35 % of the data set contains missing values, while the target variable is highly imbalanced with a ratio of 221:12. The vast majority of products were within the specifications. Prior to benchmarking misspellings were corrected. Figure 5-9 summarizes the creation of 18 production use cases from tool and die making.

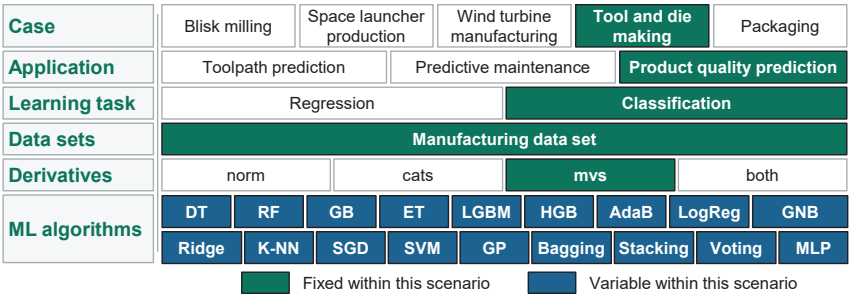


Figure 5-9: Creation of 18 production use cases from tool and die making

5.1.6 Scenario 5: Machine Tool Degradation

The last scenario is based on a publicly available data set, in which a component degradation of a shrink-wrapping machine is predicted [KAGG19]. Shrink-wrappers are used for packaging purposes. In this example, cans and bottles are packaged in three steps. Bottles and cans are first grouped, subsequently wrapped in plastic film and finally heat-shrunk. The plastic film is first fed into the machine and subsequently cut to length so that the cans or bottles can be fully packed. In this context, the cutting tool consisting of blades represents a key component, which is responsible for a proper packaging. Blades are subject to wear, which is why they need to be set-up and maintained appropriately. Monitoring the blades degradation is realized due to the acquisition of position errors. To reduce unexpected downtimes, the position error is forecasted to identify anomalies that supports in finding worn blades at an early stage. [BIRG18a]

One of six original data sets is arbitrarily chosen, which contains 2,048 rows and eight columns. Besides the timestamp, the torque information, actual position and speed are acquired from the cutting controller. In addition to the actual position and speed, the film controller provides the position error. Given these input variables, the position error of the cutting controller is defined. The learning task is regression, since it contains a continuous target variable. The standard deviation lies at 0.039. Given this data set,

new data sets are randomly created by inserting missing values and categorical features according to the routine described in Chapter 4.2.1. Through binning two numerical features randomly, two categorical features are generated (*cats*), while the missing value version (*mvs*) contains 860 missing values (6 %). Lastly, a data set was created that contains two categorical columns and 747 missing values (*both*).

In addition to regression, a data set for two-class classification is created given the continuous target variable. For that, a data discretization through binning is realized. The splitting point is a position error of 0.04 to create the two classes. According to the provided data set and its descriptions, the splitting point is set to 0.04, because the position error starts to increase progressively from this point, which indicates the leap in tool wear. Ultimately, an imbalanced data set is created with a ratio of approximately 1:9, which represents a common challenge of predictive maintenance. As for regression, both missing values and categorical features are introduced. Two categorical features and 573 missing values were created. The combination of categorical features and missing values return two categorical columns and 860 missing values. Figure 5-10 summarizes the creation of 144 production use cases from the eight data sets from the scenario machine tool degradation.

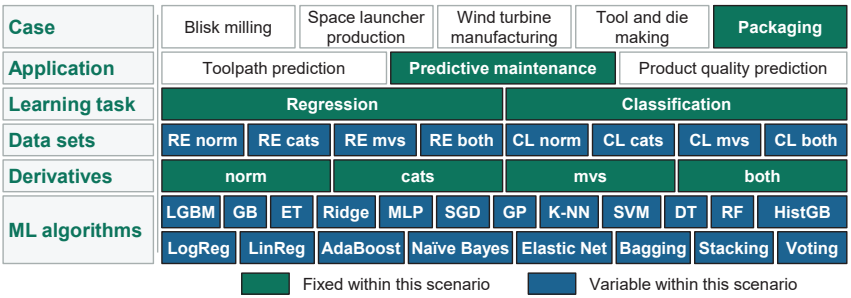


Figure 5-10: Creation of 144 production use cases from machine tool degradation

In summary, five scenarios are defined based on which Meta-DPP is validated. Table 5-1 summarizes the creation of 324 production use cases based on 22 data sets and 18 ML algorithms, from which four scenarios are stemming from past projects conducted at the Fraunhofer IPT. Both classification and regression data sets are determined to validate both meta models of Meta-DPP. In addition, derivatives are covered to validate the DPP pipelines including the imputing and encoding methods. The low quantity of rows, high dimensionality of wind turbine manufacturing compared to the number of rows, the high number of missing values and outliers in tool and die making underline the existence of data quality issues to be handled through DPP.

Table 5-1: Summary of the creation of 324 production use cases

			Number of				Existence of			
No.	Scenario	ML use case	Data sets	Use cases	RE	CL	norm	cats	mvs	both
(1)	Blisk milling	Tool path deviation	1	18	18	0	x			
(2)	Space launcher production	Product quality prediction	5	90	90	0	x			
(3)	Wind turbine manufacturing	Product quality prediction	2	54	0	36		x		
(4)	Tool and die making	Product quality prediction	1	18	0	18			x	
(5)	Machine tool degradation	Predictive maintenance	8	144	72	72	x	x	x	x
		Total	**17**	**324**	**180**	**126**	**144**	**90**	**54**	**36**

5.2 Analyzing the Results from the Case Study

Based on the five scenarios, Meta-DPP is validated. The benchmarking and pooling are validated. While benchmarking returns the actually best performing pipeline overall, pooling provides the best performing pipeline from the pool. In addition, the meta models for regression and classification are validated. Chapter 5.2.1 discusses fundamental considerations when analyzing the results of the validation of Meta-DPP. The results of the application of Meta-DPP for every scenario are analyzed in Chapter 5.2.2-5.2.6. The validation results are summarized in Chapter 5.2.7.

5.2.1 Considerations for Validating Meta-DPP

To identify the performance of Meta-DPP and to compare the results of Meta-DPP recommendations with the actual best performing pipeline overall and the best performing pipeline from the pool, different performance measures are introduced in the following (I). Secondly, a brief overview is given, which meta model implementation performs best based on the design of experiments introduced in Chapter 4.4.1 (II).

(I) Performance assessment of Meta-DPP

The performance of Meta-DPP is evaluated by assessing the benchmarking, pooling, and meta model performance. A crucial factor when identifying the performance of Meta-DPP is the determination of a baseline. Ideally, Meta-DPP is compared with the achieved performance in case no DPP is applied. Applying no DPP for the selected production use cases leads in 50 % of the cases to errors. Even though being designed to increase the performance of an ML model and thus representing a key development

within Meta-DPP, essential pipelines are used as baselines to compare the performance of the recommended output of Meta-DPP. Therefore, the benchmarking performance is determined by identifying whether the best performing pipeline outperforms essential pipelines. For assessing the performance of the pipeline pool and the meta model, three different approaches are used that are introduced in the following.

Performance assessment through DPP scores

The goal of the meta model is to recommend overall well performing DPP pipelines out of the wide range of possible DPP pipelines. To determine if the meta model is capable of identifying overall well performing DPP pipelines, the DPP scores, which were introduced in Chapter 4.2.4, are used. The idea behind this is that even if the meta model does not recommend the best performing pipeline of the pipeline pool, the model can still propose a very good performing pipeline from the pool. Figure 5-11 illustrates the validation principle based on DPP scores. The higher the DPP scores are, the better is the performance of the DPP pipeline.

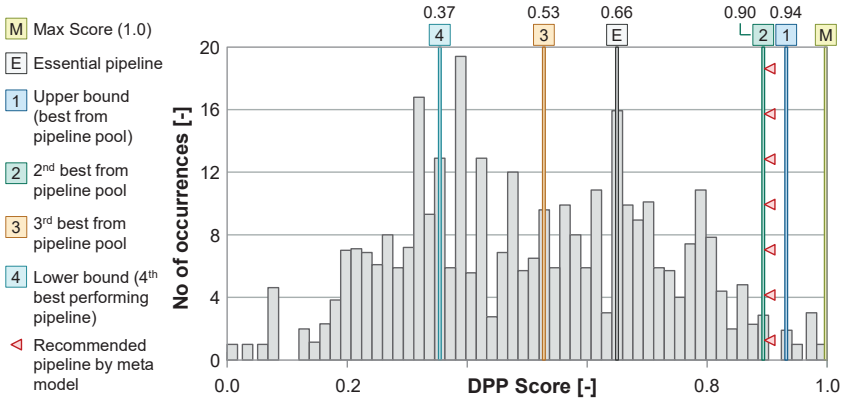


Figure 5-11: Validation principle based on DPP scores

The bar chart shows the number of pipeline occurrences over the DPP scores for a validation use case. On the far right of the chart, the maximum score (M) is shown, which represents the best possible DPP pipeline for the given use case achieved through benchmarking. The four pipelines from the pipeline pool are then mapped according to their DPP scores. In addition, the score of the essential pipeline (E) is considered, which serves as a further baseline to identify the performance of the recommended pipeline pool. Referring to Figure 5-11, the upper bound of the pool (1) exhibits a score of 0.94. The upper bound represents the pipeline with the highest DPP score

within the pipeline pool, whereas the lower bound is logically the pipeline with the lowest DPP score. Given the example in Figure 5-11, the recommended pipeline is the second best performing pipeline (2), which achieves a DPP score of 0.90. Even though the upper bound is not selected, a recommendation with a scaled rank of 0.90 means that the selected pipeline is under the top 10 % best performing pipelines for the validation use case.

The information of DPP scores achieved by the meta model can then be compared with other pipeline scores from the pool. Since the DPP scores are calculated for every use case, the mean is calculated. The maximum DPP score is achieved if the meta model selects the upper bound every time. In addition, the mean DPP score of the meta model is eventually compared with the mean scores of every pipeline from the pipeline pool and the essential pipeline.

Performance assessment through distance measures

Based on and closely related to the DPP scores, distance measures can be calculated. While the DPP scores indicate the overall performance of the pipeline pool, the distance measures show directly how close the recommended pipeline is to the best performing DPP pipeline of the pool. Four distance measures are derived, which can be taken from Figure 5-12. The 3rd best performing pipeline from the pool is used for demonstration purposes. The distances are calculated from the DPP scores and provide another representation of the performance of the meta model.

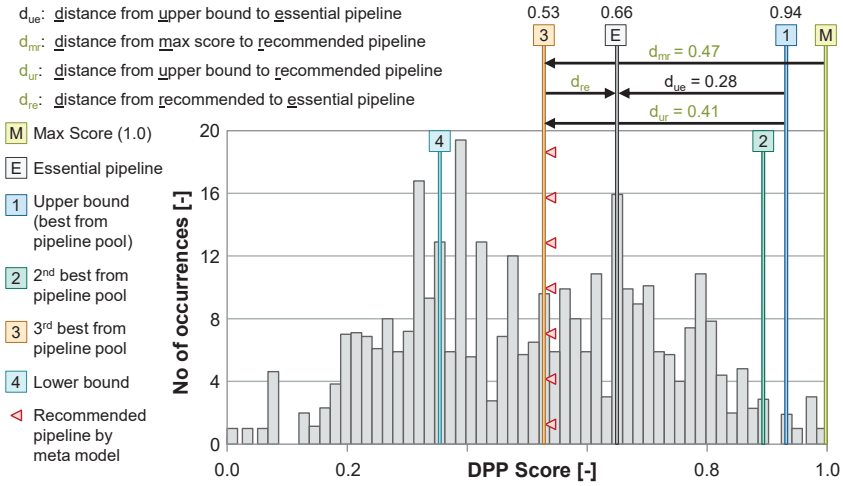


Figure 5-12: Validation principle based on distance measures

Distances that consider the recommended pipeline are written in green. The black distance d_{ue} represents the pooling performance and is computed by the difference between the upper bound and essential pipeline score. Referring to the performance of recommended pipelines, the distance from the maximum score to the recommended pipeline is calculated to obtain the information about the distance from the theoretically best performing to the recommended pipeline (d_{mr}). The distance d_{mr} is 0.47, which is the difference of the max score (1.0) and the score of the recommended pipeline (0.53). The distance between the scores of the essential and recommended pipeline d_{re} is further considered for baseline comparison. If the DPP score of the essential pipeline is higher than the score of the recommended pipeline, the distance values become negative. The distance between the upper bound of the pool and the recommended pipeline is calculated. If the meta model is able to predict the upper bound for every use case, the mean of the distance is $d_{ur} = 0$.

Meta model performance

In addition to the DPP scores and distance measures, the meta model performance is considered to determine the capability of the meta model to recommend DPP pipelines. In contrast to the previous measures, the meta model performance concentrates on the correct classification of the DPP pipeline given the pipeline pool. As stated in Chapter 2.3.2, more than one performance metric needs to be chosen to identify the performance of the meta model. For that reason, the F1 score, precision, recall and accuracy are used to determine the ability of the meta model to recommend DPP pipelines successfully. Beside these metrics, the probabilities of each class are considered to identify how certain the meta model is about its recommendation. Given the design of experiments introduced in Chapter 4.4.1, the best performing meta model is determined using the test data, i. e., 324 production use cases.

(II) Meta model

Given the design of experiments from Chapter 4.2.4 and all use cases from the case study, the best meta model is determined based on two constraints: First, the meta model performance needs to be higher than arbitrarily choosing a pipeline from the pipeline pool. Therefore, the F1 score, accuracy, precision, and recall need to be higher than 0.25. Second, the meta model is finally selected that achieves the highest median DPP scores to follow the pooling procedure from Chapter 4.2.5.

Applying the criteria for classification, 24 out of 36 configurations achieve higher scores than 0.25 for F1 score, accuracy, precision, and recall. Every of the six meta models are within the top 24 configurations. In case of classification, 30 and all meta features achieve higher performances than 20 meta features. Meta models that are trained on

data sets that do not consider pipeline-related meta features show similar performances as those that are trained on all meta features in terms of F1 score. Ultimately, an *extra trees classifier* shows the highest median DPP score on 30 meta features for classification, which do not contain pipeline-related meta features.

In case of regression, 15 out of 36 configurations achieve higher F1 score, accuracy, precision, and recall than 0.25. The *decision tree*, *extra trees*, *gradient boosting classifier* as well as *logistic regression* represent candidate meta models. The highest median DPP score is achieved by the *gradient boosting classifier* with all but pipeline-related meta features. Even though meta models that are built on no pipeline-related meta features achieve the highest DPP score, eight of the top 15 meta models rely on all meta features to make a recommendation. This results in the final settings depicted in Figure 5-13.

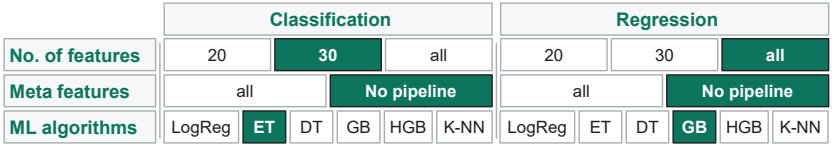


Figure 5-13: Settings of best performing meta model

Even though the *decision tree classifier* achieves higher F1 scores of 0.36 for classification with all meta features and including pipeline-related meta features, the *extra trees classifier* with an F1 score of 0.33 is chosen due to the higher median DPP scores. The *extra trees classifier* mainly relies on the ML algorithm-related meta features as discussed in Chapter 4.4.2. The *gradient boosting classifier* selected for regression both achieves highest F1 score and median DPP scores. The results of every of the 36 configurations for both classification and regression can be taken from Annex A.14. A thorough discussion of the achieved DPP scores, distances and ML model performance metrics of the best meta models is presented in the analysis of the case study results in Chapter 5.2.2 to 5.2.7.

5.2.2 Scenario 1: Blisk Milling

The results for the Blisk milling scenario containing 18 production use cases can be taken from Figure 5-14 and are discussed in the order of benchmarking, pooling and meta model performance. The upper half shows the benchmarking performance, while the lower half of Figure 5-14 represents the pooling and meta model performance.

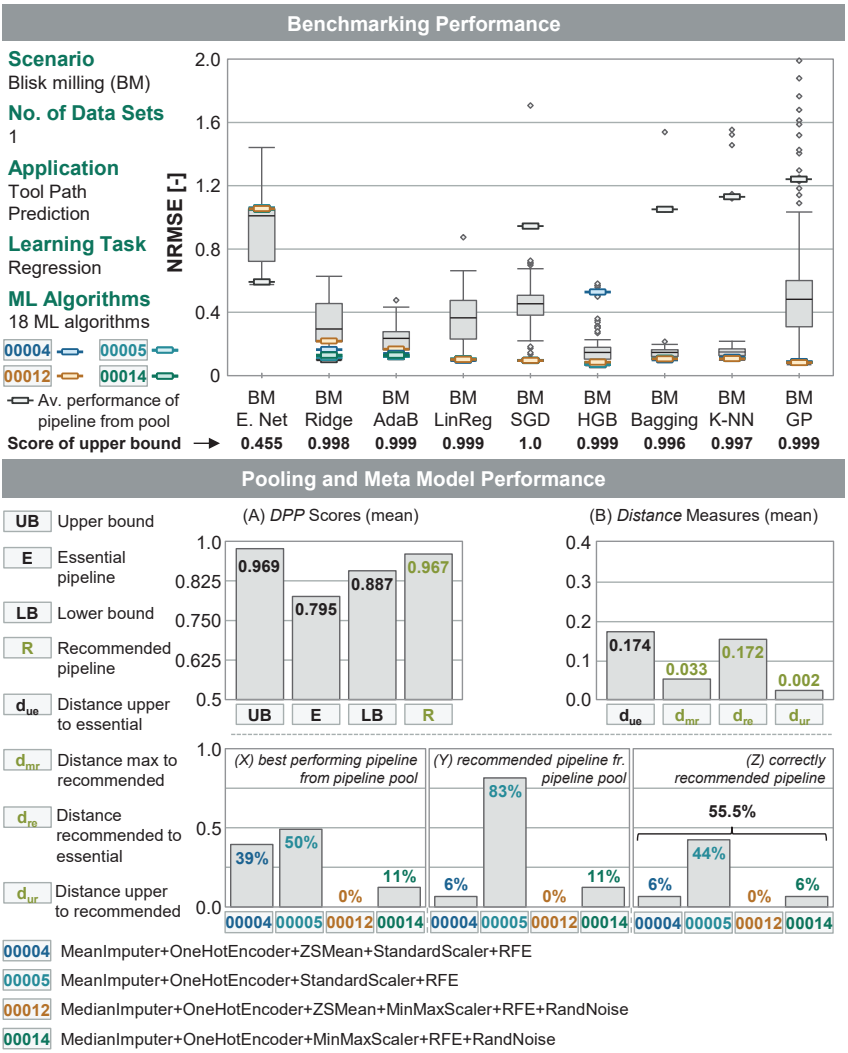


Figure 5-14: Benchmarking, pooling, and meta model results for Blisk milling

The upper half of Figure 5-14 shows achieved NRMSE values for the Blisk milling (BM) scenario based on nine exemplary use cases. The high range of NRMSE values of 0.05 to 2 confirms the observations from the benchmarking results (see Chapter 4.2.3) and underlines the high dependency between DPP pipelines and ML models. For *BM-AdaBoost*, the range is from 0.1 to 0.5, while the range is close to 2.0 for *BM-GP*, i. e., the *Gaussian process classifier* is more sensitive to DPP than *AdaBoost*. In 17 out of 18 use cases, top performing pipelines of the benchmarking outperform essential pipelines. Every essential pipeline and pipeline from the pipeline pool provide recommendations with no error. In the benchmarking graph, the performances of the pipeline pool are mapped. As already seen during the development of Meta-DPP, best performing pipelines are distributed over all four pipelines from the pool. In terms of the pool, use cases are available, in which pipelines are widely spread as for *BM-HGB* or very close together as for *BM-SGD.*

The lower half of Figure 5-14 provides details about the pooling and meta model performance and is divided in two sections. The upper section comprises the DPP score (A) and distance measure (B). Black fonts consider the pooling, green colors the meta model performance in the upper part. To compare the performances of essential pipelines and the pipeline pool, averaged scores are calculated to account for outliers in the recommendations. The upper bound score shown in (A) represents the theoretically maximum possible DPP score that can be achieved by the meta model. It has a mean score of 0.969 meaning that the pipeline pool is capable of proposing pipelines within the top 4 % given the benchmarking results on average. The upper bound outperforms the essential pipelines by a distance d_{ue} of 0.174, which can be taken from (B). In 72 % of all production use cases, the upper bound achieves higher DPP scores than essential pipelines.

The meta model outperforms essential pipelines by a distance d_{re} of 0.172, i. e., the meta model performs on average 22 % better than the essential pipeline. In addition, the DPP score of the recommended pipeline is close to the upper bound (d_{ur} = 0.002). This means that the meta model is capable of recommending DPP pipelines successfully since it outperforms essential pipelines, while being close to the theoretical maximum DPP score of 0.969. The lower section comprises the performance of the meta model. In (X), the class distribution of the actually best performing is provided, while (Y) shows the distribution of the recommended pipelines. (Z) presents the correctly recommended pipelines given the actual class distribution representing the accuracy. The meta model can recommend the actually best performing pipeline in 55.5 % of all cases. The accuracy is the sum of all correctly classified pipelines in (Z). For instance,

the pipeline with the ID 00014 is actually in 11 % of all cases the best performing pipeline (X). From this 11 %, the meta model proposes the pipeline in 50 % correctly, which yields the value of 6 % (Z). The F1 score lies at 47.2 %, while the precision is 67.8 %.

In summary, Meta-DPP is capable of providing a pipeline pool, which recommends pipelines in the top 4 % of the benchmarking results and a meta model, which outperforms essential pipelines by a distance d_{re} of 0.152 given the accuracy of 55.5 %. In addition to the discussion of the overall performance, two use cases are compared with a correct and false recommendation.

Comparison of two exemplary Blisk milling use cases (from 18 use cases)

Figure 5-15 shows a comparison of the use case *BM-AdaB* (left) and *BM-Bagging* (right). The number of pipeline occurrences is depicted over the DPP score. Both production use cases are considered in the benchmarking results in Figure 5-14, from which the NRMSE values can be taken. In case *AdaBoost* is applied, the top performing pipeline (ID 00005) is correctly recommended with a DPP score of 0.999. On the contrary, using a *bagging classifier* leads to a false recommendation (ID 00005) by the meta model, which proposes the third best performing pipeline with a DPP score of 0.989. Even though the meta model is not capable of providing the correct recommendation, the pipeline is still under the top 2 % of all pipelines. Using the *gradient boosting* as meta model results in 95 % certainty for the correct classification of ID 00005. For *BM-Bagging*, the meta model chooses ID 00005 wrongly with 93 % certainty.

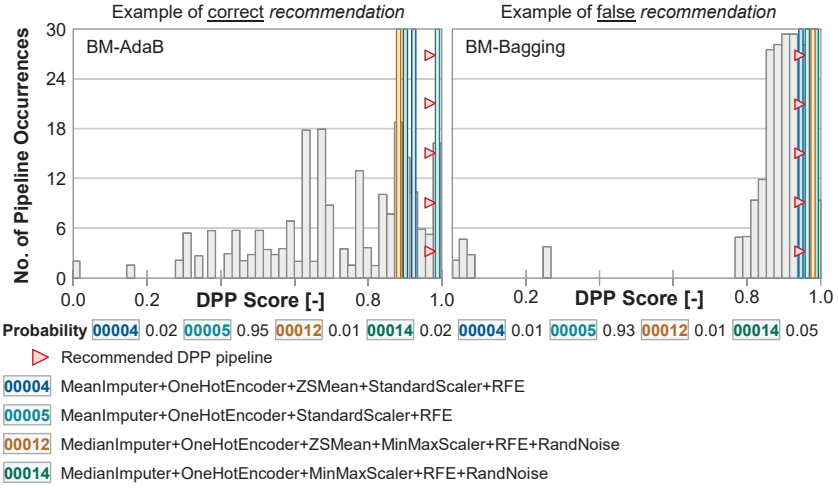


Figure 5-15: Comparison of correct and false recommendation for two use cases from Blisk milling

5.2.3 Scenario 2: Space Launcher Production

The results for the space launcher scenario including five data sets can be taken from Figure 5-16. From each of the five data sets, one to two exemplary use cases are shown in the benchmarking performance graph. Every pipeline considered for benchmarking did not throw any error. The different ranges in NRMSE values and distribution of the pipeline pool can also be observed for space launcher production. In 95 % of the 90 use cases, top performing pipelines of the benchmarking achieve lower RMSE values than essential pipelines.

Referring to the pooling results, the upper bound achieves a mean DPP score of 0.958. On average, the pipeline pool contains at least one pipeline, which is under the top 5 % of best performing pipelines. The range of the pipeline pool, i. e., the distance between the lower and upper bound score, is 0.261 and significantly higher as the Blisk milling scenario. While the pipeline pool outperforms essential pipelines with a distance d_{ue} of 0.083, the recommendation from the meta model (*gradient boosting*) achieve the same DPP score as the upper bound resulting in a distance to essential pipelines of d_{re} = 0.083. Consequently, essential pipelines outperform the meta model recommendation in 69 % of all cases. However, differences occur when looking into data sets individually. For instance, the upper bound for the data set *LH2 M1B* attains a score of 0.976, the lower bound of 0.841, and the meta model a score of 0.976, respectively. The meta model then outperforms essential pipelines by a distance d_{re} of 0.232. On the contrary, while the upper bound for the data set *UITS M1* also lies at 0.976, the lower bound achieves a score of 0.768 on average. In this case essential pipelines achieve a score of 0.918, which results in a distance to the essential pipelines of d_{re} = 0.058. For all five data sets, the pipeline pool as well as meta model outperforms essential pipelines. Ultimately, the meta model achieves an accuracy of 90.0 %. The distribution of best performing pipelines (X) from the pipeline pool in Figure 5-16 depict that in 83 % of all cases, ID 00005 is actually best performing. The meta model recommends ID 00005 in 89 % of all cases. By applying the *gradient boosting* as meta model, ID 00005 is correctly recommended in 82 % of all cases. This indicates a huge imbalance in the actual but also recommended pipelines towards pipeline ID 00005.

In summary, Meta-DPP provides a pipeline pool, which recommends pipelines in the top 5 % of the benchmarking results on average and a meta model (*gradient boosting*), which is equally performant to the upper bound. Thus, both upper bound and meta model outperforms essential pipelines by a distance d_{re} of 0.083 given the F1 score of 40.5 %.

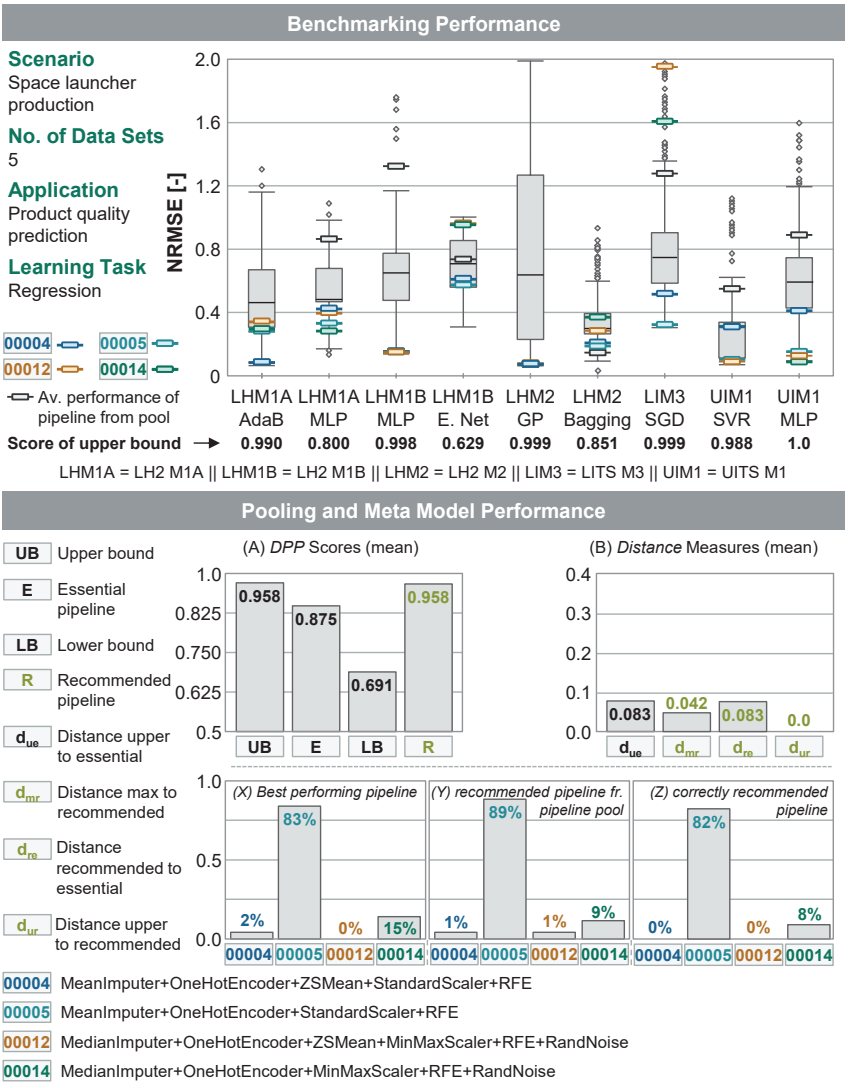


Figure 5-16: Benchmarking, pooling, and meta model results for space launcher production

Comparison of two space launcher production use cases (from 90 use cases)

The two production use cases shown in Figure 5-17 focus on the data set *LH2-M1A*. On the left, the correct recommendation of ID 00005 can be seen that achieves a DPP score of 0.998 for *LH2-M1A AdaB*. A wrong recommendation can be found on the right, in which ID 00005 is recommended while being the second best pipeline in the pipeline pool. Even though the recommendation is wrong, ID 00005 receives a score, which is only 0.025 lower than the top performing pipeline (ID 00014). This shows the necessity of also focusing on achieved scores instead of only meta model performances since the second best pipeline still provides competitive results. Even though ID 00005 is recommended, the probability is significantly less compared to the correct recommendation. ID 00014 receives the second rank with a probability of 0.30. Thus, the values indicate that the model is able to extract information from the training data although being imbalanced in recommending ID 00005.

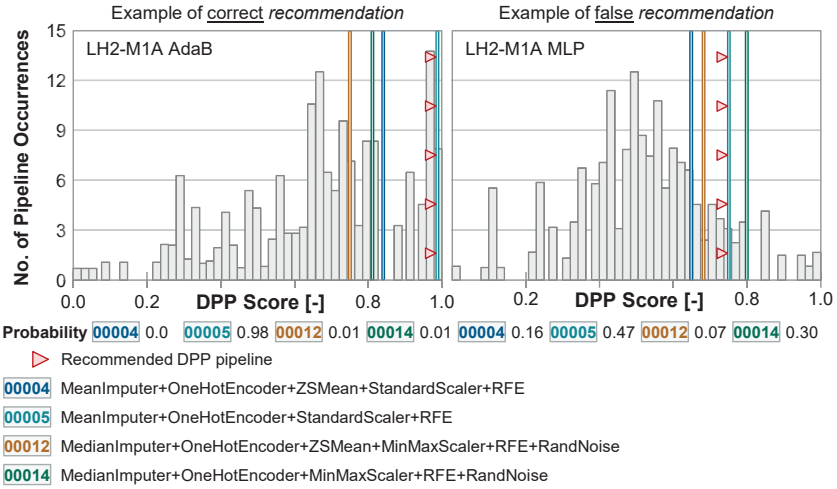


Figure 5-17: Comparison of correct and false recommendation for two use cases from space launcher production

5.2.4 Scenario 3: Wind Turbine Manufacturing

The results of wind turbine manufacturing (WTM) aggregated for two classification data sets (WTM1, WTM2) can be taken from Figure 5-18. The benchmarking performance show 9 from 36 exemplary use cases. Achieved F1 scores are maximum at 0.75. The ranges in F1 scores are between 0.2 and 0.75.

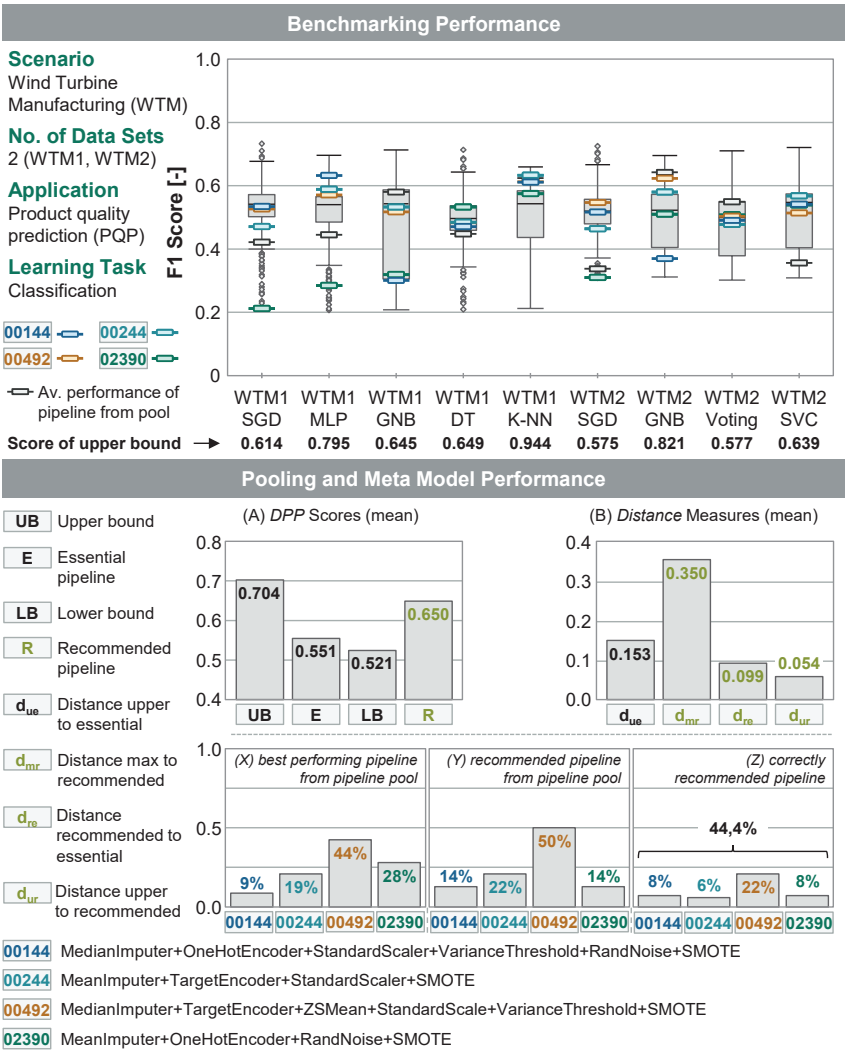


Figure 5-18: Benchmarking, pooling, and meta model results for WTM

In 100 % of all cases, best pipelines from the benchmarking achieve higher performances than essential pipelines. Every essential pipeline as well as pipeline from the pool provide recommendations with no error. Even though the performances of the two data sets WTM1, and WTM2 are comparable, the ranges in F1 scores are lower for WTM2. In addition, the comparison of the use cases *WTM1-SGD* and *WTM2-SGD* as well as *WTM1-GNB* and *WTM2-GNB* show that best performing pipelines from the pool are different, which confirms the observation from Chapter 4.2.3. Pipelines within the pool are either widely spread (*WTM1-GNB*) or close together (*WTM2-Voting*).

Referring to the pooling performance, the upper bound of the pool achieves a DPP score of 0.704 on average. The pipeline pool is therefore capable of recommending top 30 % pipelines from the benchmarking. Furthermore, the upper bound significantly outperforms the essential pipeline by a distance d_{ue} of 0.153. For the 36 production use cases, essential pipelines show worse performance in 89 % compared to the upper bound. When comparing the mean DPP score of the upper bound of WTM1 and WTM2, the pipeline pool achieves a DPP score of 0.75 for WTM1 and 0.66 for WTM2, showing the difference in performances when considering different quality tolerances. The same deviation applies for the recommended pipeline scores, which is 0.69 for WTM1 and 0.61 for WTM2.

Focusing on the aggregated performance, the meta model (*extra trees classifier*) outperforms essential pipelines by a distance d_{re} of 0.054, i. e., the meta model performs on average 18 % better than the essential pipeline. In addition, the DPP score of the recommended pipeline is close to the upper bound (d_{ur} = 0.054). The extra trees classifier is capable of recommending the actually best performing pipeline in 44.4 % of all cases, providing an improvement 77.7 % than randomly selecting a pipeline from the pool. Besides the accuracy of 44.4 %, the F1 score lies at 47.2 % and recall at 52.1 %.

As a conclusion, the benchmarking outputs best performing pipelines that outperform essentials in 100 % of all cases. The pipeline pool outperforms essential pipelines by a distance of d_{ue} = 0.153, while achieving a DPP score of 0.704 on average. The meta model achieves an accuracy of 44.4 % and outperforms essential pipelines d_{re} by 0.054.

Comparison of two wind turbine manufacturing use cases (from 36 use cases)

On the left of Figure 5-19, the use case of *WTM2-Voting* is shown with a correct recommendation. The *extra trees classifier* provide a ranked recommendation through the different probabilities for each class. ID 02390 has a probability of 0.51, followed by ID 00492 with 0.28, ID 00144 with 0.11, and ID 00244 with 0.1. Moreover, the order of the recommendation provided by the meta model matches the actual order of pipeline performance. The DPP score achieved by the meta model is 0.577. The right of Figure

5-19 provide details of an entirely wrong recommendation by the meta model. Given the use case *WTM2-SGD*, the pipeline with ID 02390 is chosen that has a DPP score of 0. The order of the recommendation is reciprocal to the actual performance of the pool.

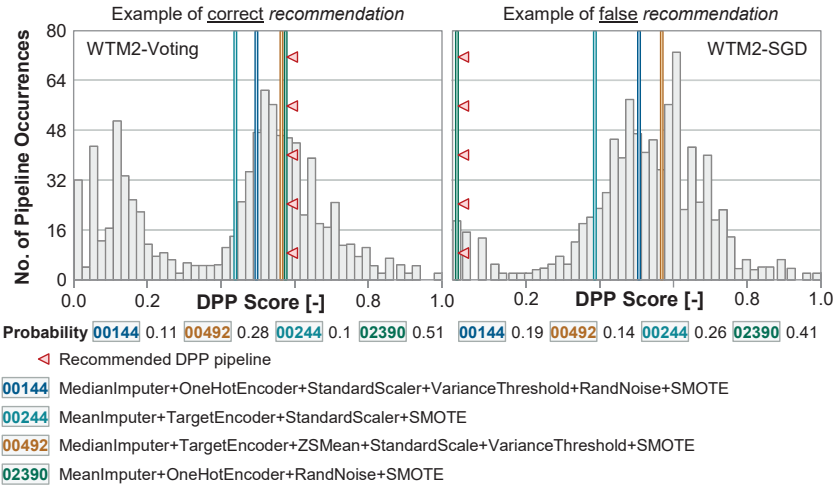


Figure 5-19: Comparison of correct (left) and false (right) recommendation for two representative use cases from wind turbine manufacturing

5.2.5 Scenario 4: Tool and Die Making

The results of the benchmarking, pooling, and meta model performance for the tool and die making scenario can be taken from Figure 5-20. Given nine out of 18 representative use cases, the F1 scores range from a minimum of 0.05 in case of *TDM SVC* to a maximum of 0.81 for *TDM HGB*. No error occurred during benchmarking.

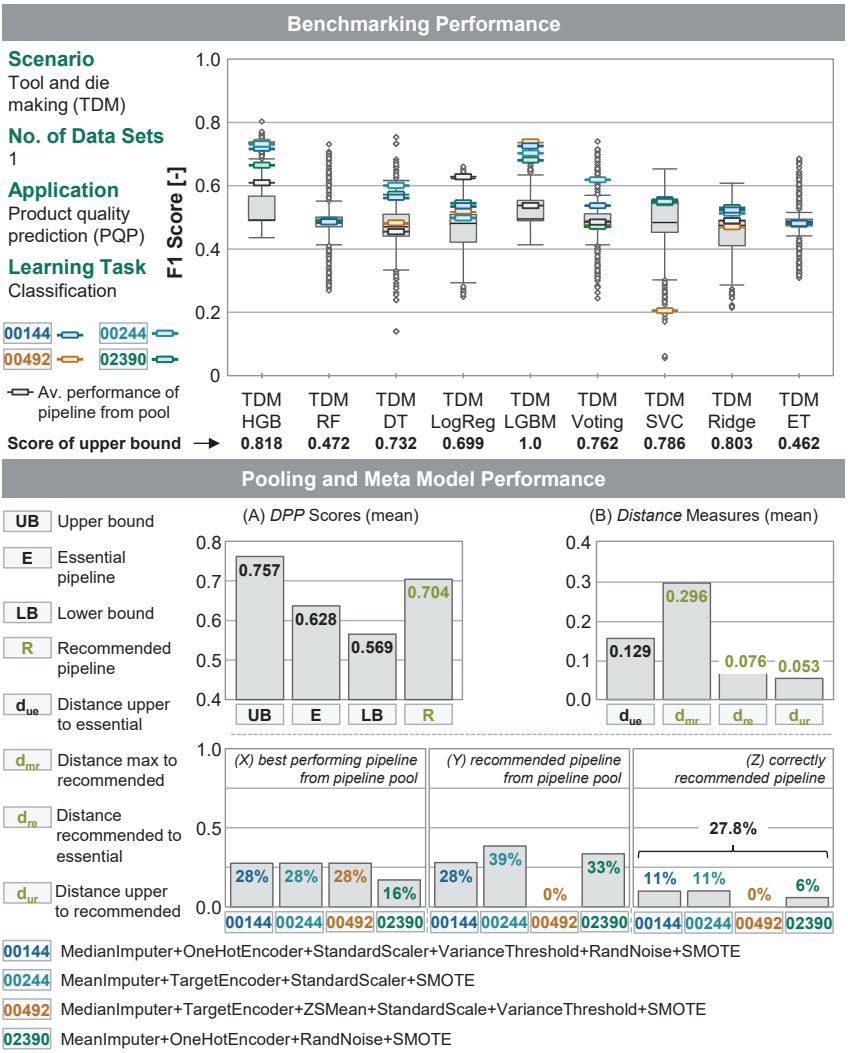


Figure 5-20: Benchmarking, pooling, meta model performance for tool and die making

Best performing pipelines achieved higher F1 scores than essential pipelines in every of 18 use cases. The pooling results reveal that the pipeline pool outperforms the essential pipelines by a distance d_{ue} of 0.129. While the pipeline ID 00492 achieve the highest DPP score of 1 for *TDM-LGBM*, the pipeline also attains the lowest score in case of *TDM-ET*, which lies at 0.458. This confirms the *no free lunch theorem*, which has been observed in the development of Meta-DPP. In 14 of 18 cases, the pipeline pool outperforms the essential pipelines. The meta model proposes the correct pipeline in 5 out of 18 use cases leading to an accuracy value of 27.8 %. This results into the distance to the essential pipelines of d_{re} = 0.076, i. e., the recommended pipelines achieve an increase by 12 %. Further values can be obtained from Figure 5-20.

Comparison of two tool and die making use cases (from 18 use cases)

Figure 5-21 show an example for of a correct recommendation by the meta model for *TDM-Ridge*. The *extra trees classifier* show a certainty of 0.73, however, proposes to apply the worst performing pipeline in the pool as second option. The actually second best pipeline in the pool (ID 00144) receives a probability of 0.04. On the right, a false recommendation is shown for *TDM-LogReg*, in which the pipeline ID 00244 is chosen that has a DPP score of 0.542. However, the best performing pipeline with ID 02390 achieves a score of 0.699 and is recommended second with a probability of 0.35. Although not providing a correct recommendation, the meta model still achieves a reasonable score and by using the probability, best performing pipelines are found.

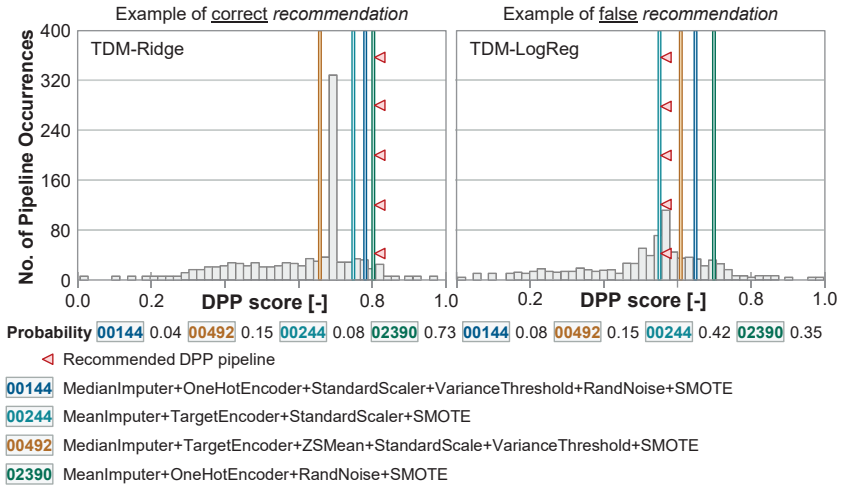
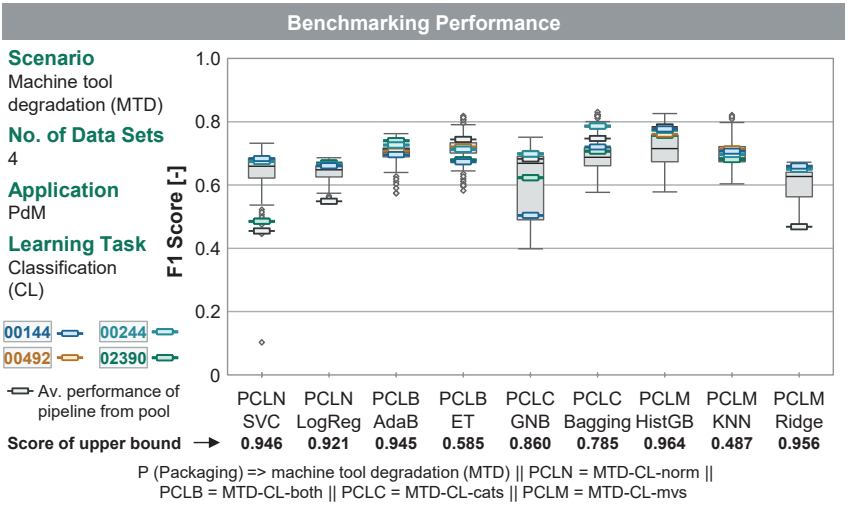


Figure 5-21: Comparison of correct (left) and false recommendation (right) for two representative use cases from tool and die making

5.2.6 Scenario 5: Machine Tool Degradation

At first, the classification results of machine tool degradation (MTD) are analyzed followed by regression. Figure 5-22 shows an excerpt of nine of 72 use cases of the four data set derivatives of *norm, both, cats, mvs*.



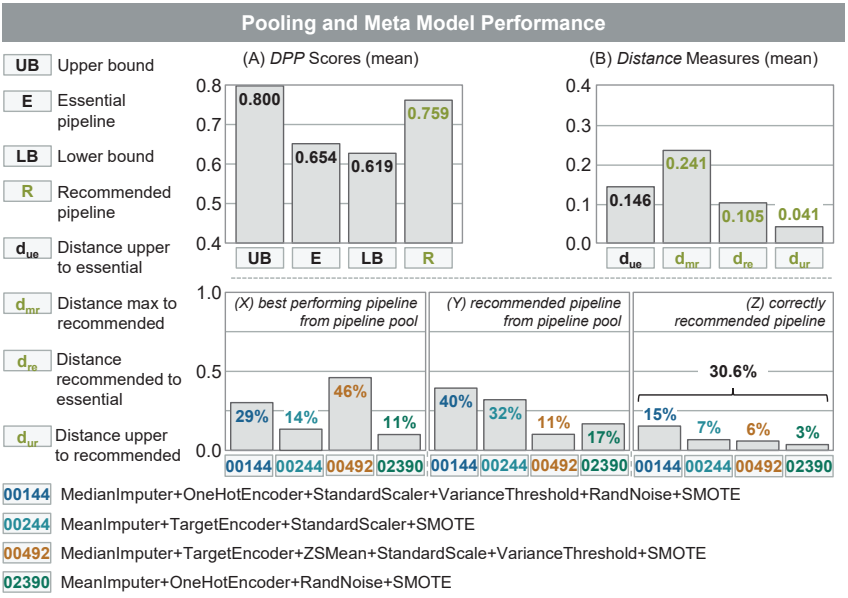


Figure 5-22: Benchmarking, pooling and meta model performance for MTD (CL)

Since the MTD scenario is based on a packaging machine, the abbreviation used for discussing the use cases is *P*. With the exception of *PCLN SVC* the F1 scores range between 0.4 and 0.83. The pipeline pool outperforms the essential pipelines with a distance of d_{ue} of 0.146. The recommended pipeline by the meta model outperforms essential pipelines by the distance d_{re} of 0.105. The meta model achieves an accuracy of 30.6 %. Even though representing similar benchmarking performances, the scores of the pipeline pool differ depending on the derivative. While for *norm* data set, the DPP score of the pipeline pool lies at 0.750, the pool achieves a score of 0.828 for *both* and outperforming the essential pipelines by a distance of d_{ue} of 0.154. For *cats*, the pool achieves the highest score of 0.841, while *mvs* achieves 0.776. The distance measures of the recommended to essential pipeline reveal a similar behavior. The highest distance d_{re} is achieved in case of the derivative *both* with 0.160 and lowest with a distance of d_{re} = 0.076. The difference can be explained by averaging the additional essential methods of imputing and encoding, which are applied in case of *both*.

Even though the performances differ between data sets, in summary, the benchmarking results in best performing pipelines that achieve higher F1 scores in 100 % of 72 use cases. The upper bound as well as meta model outperform essential pipelines by a distance d_{ue} of 0.146, and a distance d_{re} of 0.105. Ultimately, the meta model achieves a F1 score of 28.5 %.

Comparison of two machine tool degradation classification use cases (from 72 use cases)

Focusing on two particular use cases for the derivative *both*, the correct recommendation can be seen on the left in case *Both-AdaB*. The pipeline ID 02390 achieves a DPP score of 0.945. The probability lies at 0.44 followed by a probability of ID 00144 of 0.25 showing the worst performance. For the false recommendation, the probability for ID 02390 is even higher in case of *Both-DT* that lies at 0.59. Here, the achieved DPP score is only 0.459. In these examples, the choice of the correct pipeline accounts for a difference in DPP score of 0.3. Since the meta model can provide wrong recommendations, the pipeline pool can be used to try out the pipelines according to the probability.

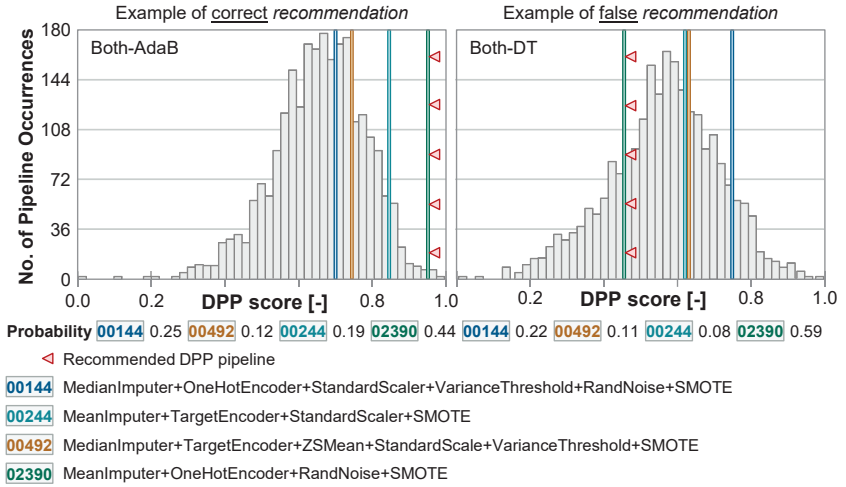


Figure 5-23: Comparison of correct (left) and false recommendation (right) for two representative use cases from machine tool degradation (classification)

Regression (RE)

The achieved NRMSE values for production use cases from four data set derivatives of machine tool degradation can be seen in the upper half of Figure 5-24. The best pipelines achieve NMRSE values of 0.42.

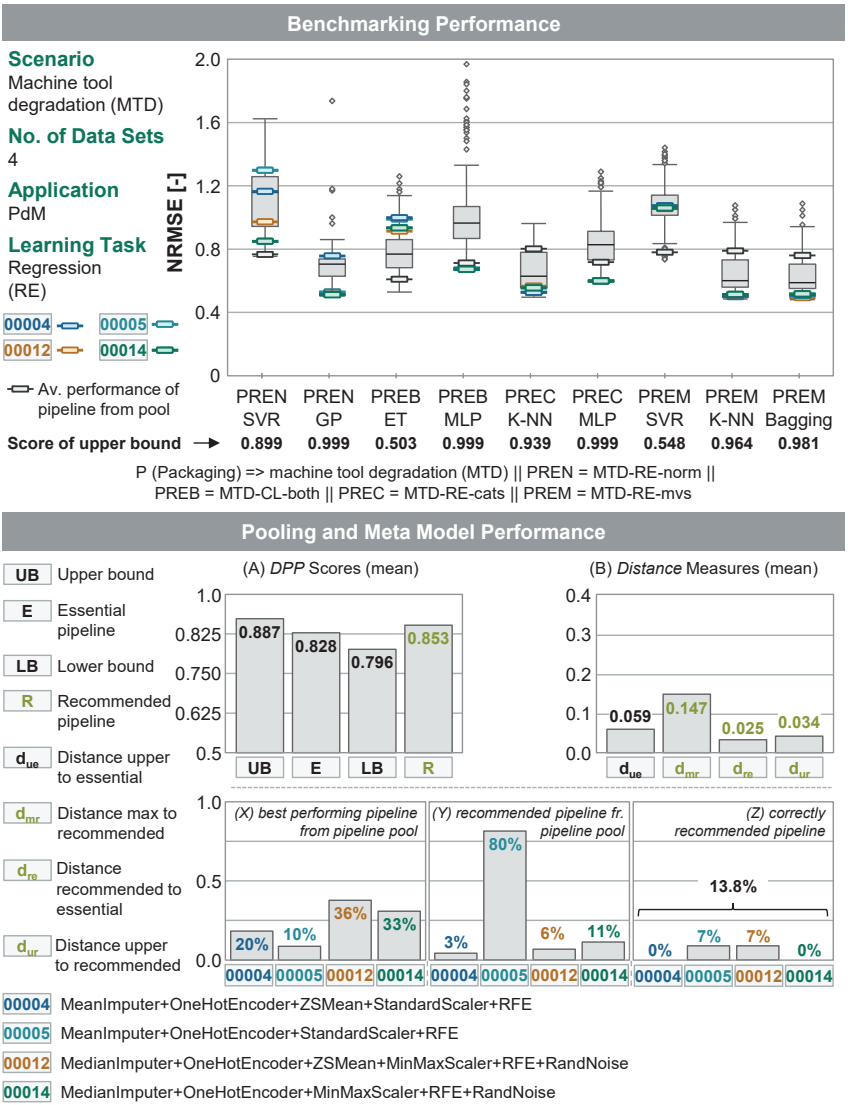


Figure 5-24: Benchmarking, pooling, and meta model performance for MTD (RE)

Pipelines from the pool achieve a mean DPP score of 0.887, while the essential pipeline achieves a score of 0.828. The range of the upper and lower bound lies at 0.091 and is low compared to the ranges of previous scenarios. On average, the upper bound outperforms essential pipelines with a distance d_{ue} of 0.059. The meta model slightly outperforms the essential pipelines by a distance of d_{re} of 0.025, resulting in a performance increase of 3 %. However the accuracy of the meta model only lies at 13.8 %. The meta model recommends in 80 % of all 72 use cases pipeline ID 00005, which only is represented in 10 % of all use cases as best performing pipeline from the pool.

Different results are observed referring to the data set derivatives. In three out of four data sets, the pipeline pool outperforms essential pipelines. For the *norm* data set, the DPP score is 0.922 outperforming essential pipelines by d_{ue} of 0.108. Similar results are achieved for the derivatives of *cats* and *mvs*. The derivative *both* represents an outlier with a DPP score of the pipeline pool of 0.806, while the essential pipelines achieve a DPP score on average of 0.886. When comparing the recommended with essential pipelines, the recommended pipelines achieve higher scores by a distance of d_{re} = 0.066 for *norm* and *mvs* and 0.078 for *cats,* respectively. As for the upper bound of the derivative *both*, the recommended pipeline is outperformed by essentials with a distance d_{re} of -0.169. For *MTD-norm*, the meta model (*gradient boosting*) achieves an accuracy of 27.8 %, while for *MTD-both* and *MTD-cats*, the accuracy lies at 5.6 %. The imbalance in recommending ID 00005 can be observed for every data set derivative.

Comparison of two machine tool degradation regression use cases (from 72 use cases)

The correct recommendation for the use case *Both-GP* shows a mean DPP score of 0.987 with a probability of 0.48 (see Figure 5-25). In general, the pipeline pool achieves high scores for the *GP classifier*. The worst performance by ID 00004 is represented with a score of 0.916. For *Both-ET*, ID 00005 is chosen with a probability of 0.84. However, ID 00014 is actually best performing for this use case. As already seen within other scenarios, the DPP score of the recommended pipeline is 0.103 lower than the top performing pipeline from the pool. This underlines the importance of applying the DPP score measures in addition to the meta model performances.

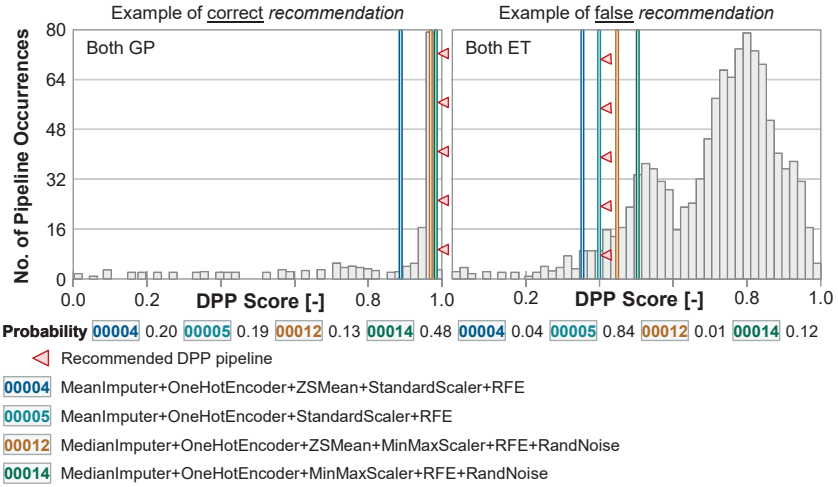


Figure 5-25: Comparison of correct (left) and false recommendation (right) for two representative use cases from MTD (regression)

5.2.7 Summary of Validation Results

In the following, the validation results are summarized starting with classification. Every result will be discussed in the order of benchmarking, pooling and meta model performance.

Classification

The classification results can be taken from Figure 5-26. Considering every classification scenario, seven data sets, i. e., 126 production use cases were used for validating Meta-DPP. In the upper half of Figure 5-26, the benchmarking performance is shown based on nine representative results of the use cases. From every of the seven data sets at least one use case is covered. The achieved F1 scores indicate the benchmarking performance, which range from 0.08 to 0.89. In more than 99 %, top performing pipelines of the benchmarking outperform essential pipelines. In the benchmarking graph, the performances of the pipeline pool are mapped. As already seen during the development of Meta-DPP, the distribution of best performing pipelines are distributed over all four pipelines from the pool. In terms of the pool, use cases are available, in which the range is high as for *WTM2-Gaussian* or very close together as for *TDM ET*.

In the lower half of Figure 5-26, the pooling and meta model performance are depicted. The mean DPP score of the upper bound is 0.767 and median of 0.773 meaning that the pipeline pool is capable of proposing pipelines within the top 24 % given the benchmarking results. The upper bound outperforms the essential pipelines by a distance d_{ue} of 0.126, which can be taken from (B). In 71 % of all production use cases, the upper bound achieves higher DPP scores than essential pipelines. The meta model outperforms essential pipelines by a distance d_{re} of 0.099, i. e., the meta model performs on average 16 % better than the essential pipeline. In addition, the score of the recommended pipeline is close to the upper bound (d_{ur} = 0.049). The meta model is capable of recommending the actually best performing pipeline in 34 % of all cases, providing an improvement of 37 % compared to arbitrarily choosing a pipeline from the pool.

The performances between the individual cases and data sets differ from each other. While for wind turbine manufacturing, the upper bound is 0.704 with a distance to the essential pipelines d_{ue} of 0.153, the upper bound score in case of tool and die making lies at 0.757 with d_{ue} equals to 0.129. Machine tool degradation provides four data sets, in which the aggregated upper bound is 0.800 with a distance of d_{ue} = 0.146. Referring to the meta model performance, the accuracy for WTM1 and WTM2 lie at 44.4 %, for tool and die making at 27.8 % and in aggregated form for machine tool degradation at 30.6 %. For all data sets but one, the meta model shows better performances than arbitrarily choosing one of the four pipelines in the pipeline pool.

In summary, Meta-DPP provides a benchmarking and a pipeline pool for classification that outperforms essential pipelines as well as a meta model, which is capable of finding overall well performing pipelines from the pool.

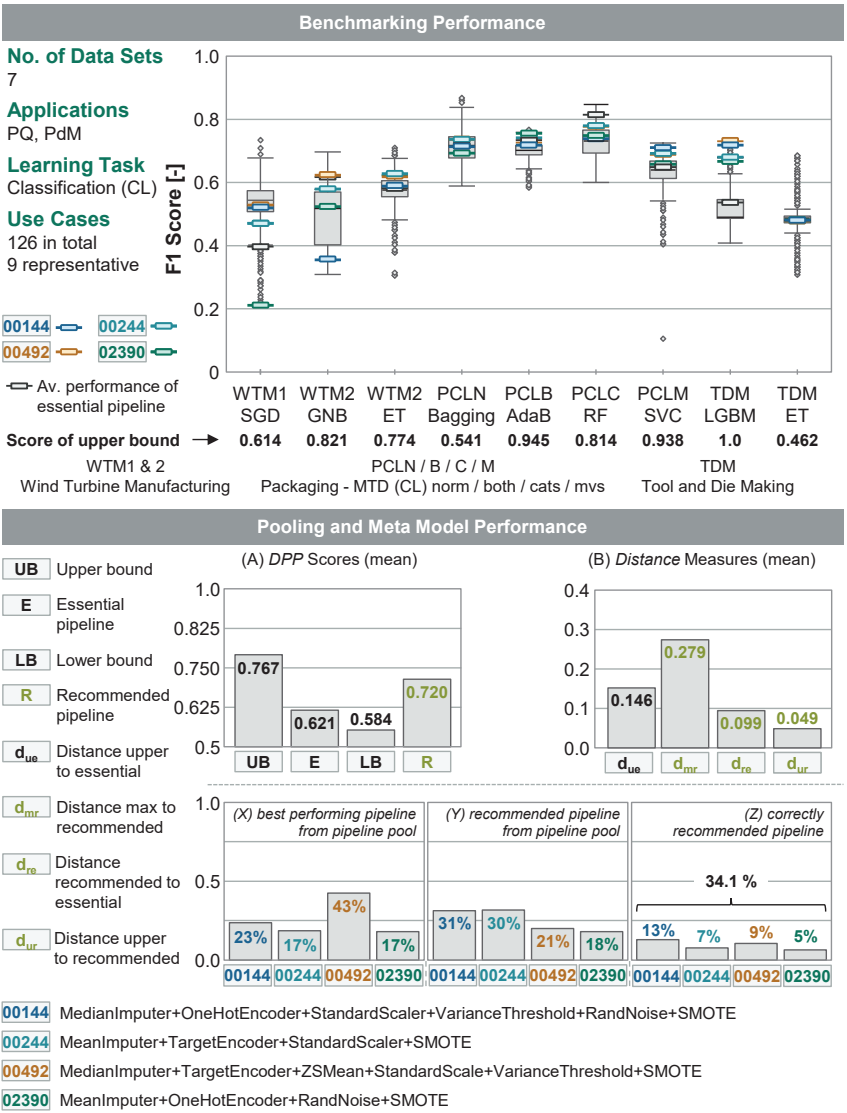


Figure 5-26: Benchmarking, pooling and meta model performance for classification

Regression

Figure 5-27 shows the results for regression, which are based on 180 production use cases given 15 data sets. The benchmarking results in the upper half of Figure 5-27 show that NRMSE values are widely spread. In general, Blisk milling shows overall performant results with NRMSE values lower than 0.5, while the data set derivatives of machine tool degradation do not show better results than NRMSE of 0.4. For the space launcher production, NRMSE values range from 0 to 2. Within benchmarking, essential pipelines represent the best performing pipeline in 2.8% of all cases. Different pipelines from the pipeline pool are best performing over 180 use cases. For *BM-AdaB*, the pipelines are close together and very close to the best performing pipeline. In case of *LH2 M2-GP*, the pipelines from the pipeline pool are spread between 0.015 and 0.999.

Referring to the lower half of Figure 5-27, the upper bound has a mean DPP score of 0.926, i. e., the pipeline pool is able to recommend the top 8 % of all pipelines being benchmarked. Essential pipelines achieve a mean DPP score of 0.829 resulting in a distance d_{ue} of 0.097. For regression, the upper bound outperforms essential pipelines in 65 % of all cases. A huge difference in performance can be identified between the individual cases. While for the Blisk milling and space launcher production scenario, the DPP score of the upper bound is 0.969 and 0.946 respectively, the machine tool degradation scenario show upper bound scores of 0.887. In all but one data set, the pipeline pool outperforms essential pipelines. The exception is represented by *MTD both*, which has an upper bound of 0.806 resulting in a distance to essential pipelines of d_{ue} = -0.169.

When applying the meta model, recommended pipelines achieve a mean DPP score of 0.911 marginally outperforming essential pipelines by a distance d_{re} of 0.079 over all use cases. For Blisk milling, d_{re} is equal to 0.172, while for machine tool degradation, it lies at 0.059 highly influenced by the negative distance through *MTD both*. Space launcher production reveals a distance d_{re} of 0.083. The DPP score of the recommended pipeline is equal to the upper bound (d_{ur} = 0.0).

When determining the performance of the meta model, correctly recommended pipelines are considered (Z). The meta model achieves an accuracy of 56.4 %. Precision lies at 38,5 % and F1 score at 31.0 %. For the Blisk milling scenario, the accuracy is 55.5 %, for space launcher production 90.0 % and for machine tool degradation at 13.9 %. In conclusion, Meta-DPP provides a pipeline pool for regression that outperforms essential pipelines and a meta model, which is capable of finding overall well performing pipelines from the pipeline pool. Regression results show a better perform-

ing pipeline pool than the classification pool and better meta model performance. However, the meta model for regression shows a high imbalance towards the class of ID 00005.

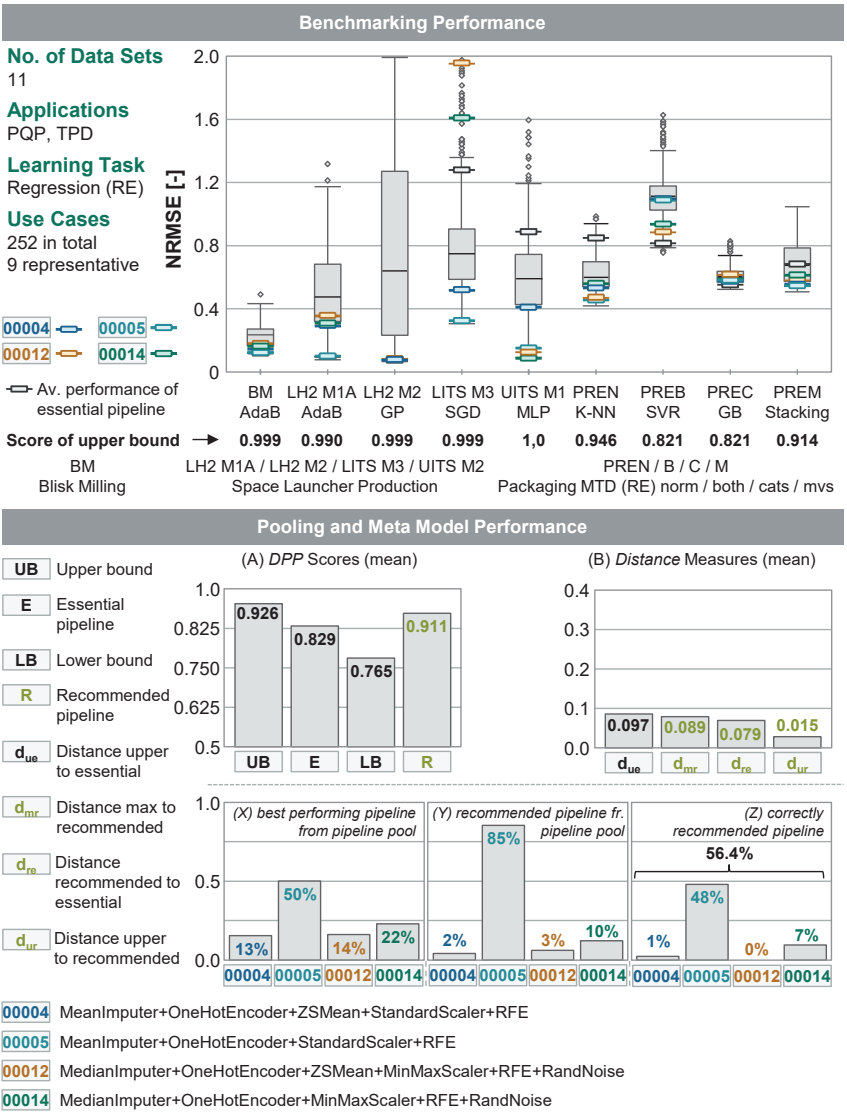


Figure 5-27: Benchmarking, pooling and meta model performance for regression

5.3 Verification

Besides the validation, Meta-DPP is verified by performing execution and special input testing in Chapter 5.3.1. Requirements that are placed on Meta-DPP were derived in Chapter 3.1. The fulfillment of these requirements is assessed in Chapter 5.3.2.

5.3.1 Execution and Special Input Testing

Execution and special input testing are performed based on the best performing meta models as well as pipeline pools for regression and classification. While execution testing aims at revealing errors when executing Meta-DPP, special input testing uses techniques to evaluate the accuracy of Meta-DPP in case various inputs are given to the system. [BALC98, pp. 366-372]

Execution testing

The appropriate execution of Meta-DPP is tested by randomly providing production use cases via the GUI. The inputs required are the data set, learning task, ML algorithm, and the explainability specification. Meta-DPP is tested on 100 random inputs that are manually inserted by a user. The test results show that for every case, Meta-DPP is capable of providing the pipeline pool including a ranked recommendation. The rule-based meta target selector is thereby verified. Since no error messages occurred and the pipeline pool was correctly chosen according to the learning task, Meta-DPP is deemed as successfully verified and validated with regard to its execution behavior.

Special input testing

The special input testing for Meta-DPP considers invalid as well as extreme input testing [BALC98, pp. 370-372]. Three types of tests are performed: wrong input, missing input, and explainability input resulting in six scenarios that are introduced in the following. First, the functionality of Meta-DPP is checked if ML algorithms and explainability specification are missing given an error-free but random input of data set and learning task. The second type comprises three tests of wrong inputs. Lastly, the functionality of the explainability input is checked. Table 5-2 summarizes the special input testing scenarios.

Table 5-2: Summary of six special input testing scenarios

	Miss.Inp.	Wrong.Inp.1	Wrong.Inp.2	Wrong.Inp.3	Expl.Inp.1	Expl.Inp.2
Data set	Classification	Classification	Regression	Non-conform	Classification	Classification
Learning Task	Classification	Regression	Classification		Classification	Classification
ML algorithms		RF	RF		RF	RF
Explainability		no	no		no	yes

Miss.Inp. Missing input testing Wrong.Inp. Wrong input testing Expl.Inp. Explainability input testing

(1) Missing input scenarios

If no input is provided by the user regarding an ML algorithm and explainability, Meta-DPP provides a recommendation in both cases by setting a *random forest* and *no explainability* as default parameters. This scenario was tested for regression and classification data sets.

(2) Wrong input scenarios

According to Table 5-2, three scenarios are checked, which depend on the combinations of a data set and a learning task. First, the user can insert a classification data set, but checks regression as being the learning task (*Wrong.Inp.1*). Meta-DPP then informs the user by the warning that only *m* targets are found in the data set:

"Warning: only *m* targets detected – Are you sure regression is the correct task?"

The output *m* represents the number of distinct values *m* found in the target variable. This approach is transferred to the second wrong input: If a regression data set is entered, the learning task classification is chosen by mistake (*Wrong.Inp.2*). The warning information, which Meta-DPP provides, is:

"Warning: over *m* targets detected – Are you sure classification is the correct task?"

In both cases, Meta-DPP provides an output based on the selected learning task. A non-conform data set is inputted in the third case, which may only contain two columns (*Wrong.Inp.3*). For these cases, Meta-DPP cannot extract meta features, which is why the warning is displayed:

"There are not enough columns in this data set, please select another."

(3) Explainability input scenarios

Since the DPP pipeline pool does currently not contain a *PCA* as DPP method, a further test is carried out to test if the explainability input is correctly processed. As *PCA* is deemed unexplainable, the pipeline pool is randomly changed by inserting DPP methods of *PCA*. In case no explainable methods are required, pipelines that include

PCA are successfully listed in the results sheet (*Expl.Inp.1*). If the user requires explainable methods to be in the pipeline pool, pipelines with *PCA* are filtered (*Expl.Inp.2*).

The probabilities of the pipelines are correctly recalculated, if the probabilities from the pipeline pool are redistributed to achieve a recommendation score sum of 1. By randomly inserting DPP methods, the functionality test also proved the extendability of the pipeline pool to more pipelines. All special input testing results can be taken from Annex A.14.

5.3.2 Assessment of Requirements Fulfillment

In the following, every requirement from Chapter 3.1 is captured and assessed, how Meta-DPP fulfills these requirements. 16 requirements were defined in three subcategories, namely fundamental prerequisites, inherent functionality and user-specific requirements. Unless otherwise specified, the requirements are to be considered as entirely fulfilled for better readability. The assessments are used to summarize the degree of fulfillments in Figure 5-28.

Fundamental prerequisites of DSS

The first requirement comprises *recommending DPP pipelines considering combinations of DPP methods*. Meta-DPP contains pipelines of a variable number of DPP methods. Thereby, the number of methods can consist only of one DPP method or multiple DPP methods as conceptualized in Chapter 4.2.2 and implemented in Chapter 4.2.3. In total, 21 DPP methods constitutes the pipelines within Meta-DPP. They are configured from the preprocessing steps of data cleaning, transformation, reduction as well as augmentation and balancing. Depending on the learning task and occurrence of missing values and categorical features, the number of DPP pipelines range from 304 to 5,472 pipelines. These pipelines can comprise only one but up to nine DPP methods, while considering every DPP category and step. Therefore, the requirement is deemed as *completely fulfilled*.

Meta-DPP is capable of *handling conditionalities* and takes into account the impact of DPP on ML model performance. This requirement is addressed in the design decisions of Meta-DPP. The category of feature scaling is set right after the essential methods of imputing and encoding to account for conditionalities in case of *PCA* and *VarianceThreshold*. In addition, the DPP methods are combined meaningfully. For that reason, augmentation is performed after reduction of features to account for computing times. Encoded features are put aside and not be used during further transformation and reduction to be less prone to noise. The power transformer is selected before data reduction and augmentation since it does not affect the data distribution. Moreover,

SMOTENC is used for encoded features to preprocess the different data scales reasonably. The requirement is therefore considered as *entirely fulfilled*.

Achieving high overall performance is attained since top performing pipelines from benchmarking outperformed essential pipelines in 100 % of all cases in the development phase and 98 % in the validation phase as described in Chapter 4.2.3. When applying Meta-DPP, the pipeline pool outperforms essential pipelines by a distance d_{ue} of 0.146 on average with a mean DPP score of 0.767 for classification. For regression, the mean DPP score lies at 0.926, while the pipeline pool outperforms essential pipelines by a distance d_{ue} of 0.097 on average (see Chapter 5.2.7). Since best performing pipelines outperformed essential pipelines in the development phase by 100 % as well as 98 % in the validation phase, respectively, and the distances d_{ue} for both classification and regression are positive in the application, the requirement is *entirely fulfilled*.

To comply with the requirement of considering *representative production use cases for developing DSS*, 43 publicly available and six data sets from past projects are used. Production use cases can be assigned to multiple applications discussed in Chapter 2.1. The number of data sets were further increased by creating data sets with categorical features and missing values, named data set derivatives, leading to 104 data sets. In addition, production use cases are created based on 18 commonly used ML algorithms in production leading to 1,872 production use cases being considered. 1,278 of these use cases fall under classification, while 514 are assigned to regression. Compared to the number of use cases of up to 300, on which systems are developed and validated in literature (see Chapter 2.5.2), the data basis is deemed as large enough. In addition, the data sets used for development exhibits a wide range of data set characteristics. First, derivatives with both missing values and categorical features are considered. The data sets also differ in terms of number of attributes, instances and data quality issues such as a low ratio of instances to attributes, outliers, or class imbalances (see Chapter 4.2.1). In total, the requirements derived as fundamental prerequisites are *completely fulfilled.*

Inherent functionality of the DSS

Meta-DPP needed to be designed to realize c*omplexity reduction*. The concept of meta learning is chosen over AutoML systems due to its lower complexity. Within the Meta-DPP development, care was taken to select only from explainable, i. e., less complex, meta models. Referring to the implemented Meta-DPP, only four inputs need to be provided by the user: data set, learning task, ML algorithm, and explainability information (see Chapter 5.3.1). In addition, the number of recommended pipelines is four in the current Meta-DPP setting, which provides the user a clear overview. The number

of pipelines can further be reduced to two pipelines if required as described in Chapter 4.2.5, since the number of pipelines in the pool represents a hyperparameter of Meta-DPP. Because the number of DPP methods is inherently minimized by providing shorter pipelines in case of equal performances, the requirement is deemed *entirely fulfilled*.

Production use cases can be very different in characteristics such as number of features or instances that require DPP pipelines with different properties to address a wide range of use cases. Therefore, *handling of use cases' variability* is reached through the creation of production use cases based on data sets and ML algorithms consisting of different properties (see Chapter 4.2.1). In addition, the selection of suitable DPP pipelines considers the criterion of DPP methods heterogeneity within a category. A benchmarking at small scale was further performed to avoid selecting multiple similar DPP methods from one category according to Chapter 4.2.2. Consequently, the requirement is *entirely fulfilled*.

The production further places domain-specific requirements. Unlike the majority of developed systems (see Chapter 3.2), both classification and regression tasks need to be handled. Meta-DPP is thus designed for *coping with production-specific requirements* by considering the F1 score as performance metric for classification and RMSE for regression as being particularly suited for unevenly distributed data sets in production. These performance metrics are used to identify best performing pipelines within benchmarking and therefore build the basis for the scaled ranking and pooling (see Chapter 4.2.3 - 4.2.5). Meta-DPP thus *entirely fulfills* the requirement. In practice, a *wide variety of ML algorithms* is applied due to the *no free lunch theorem* and high complexity of production use cases. By considering the different criteria in production (see Chapter 4.2.1), 18 ML algorithms are selected and considered within Meta-DPP, which results in the *complete fulfillment* of this requirement.

The *updatability and extendability* is manageable within benchmarking introduced in Chapter 4.2.3 and within the trained version of Meta-DPP (Chapter 4.4.2). Furthermore, the meta features database can also be updated and extended by inserting new data set-, ML algorithm-, and DPP pipeline-related meta features. Meta-DPP is designed to easily exchange or add DPP methods as well as ML algorithms within benchmarking and the creation of DPP pipeline- and ML algorithm-related meta features (Chapter 4.3). Since these extensions have successfully been performed for benchmarking and the meta features database during the development of Meta-DPP, the requirement is considered as *completely fulfilled*.

An update or extension further requires a *retraining capability* while operating Meta-DPP. As conceptually shown, the retraining of Meta-DPP is two-folded (see Chapter 4.4.2). The meta model is trained based on the actually best performing pipeline from the pipeline pool in the first stage. If the pipeline pool changes so that new pipelines are in the pool or the number of pipelines in the pool is managed, the meta features database and meta target are updated based on which the meta model is retrained. Based on the concept, the retraining can be realized. For these reasons, the *retraining capability* is deemed as being *partially fulfilled*.

Lastly, *uncertainty handling* is realized through the averaging of performances of benchmarking to reduce the randomness. Furthermore, the probabilities of the meta model indicate of how certain the meta model is about a recommendation. Further aspects of uncertainties such as the calculation of confidence intervals or sensitivity analyses could be used to additionally determine the uncertainty of Meta-DPP. Individual core components of Meta-DPP such as the benchmarking of the meta target selector, the pooling or the meta model can further be investigated with regard to uncertainty. Ultimately, handling uncertainty is *partially fulfilled*.

User-specific requirements

Meta-DPP provides recommendations based on four inputs. To enable the target group, also consisting of non-programmers, in using Meta-DPP, a user interface is created. The user is enabled to systematically describe the production use case based on the inputs of data set, learning task, ML algorithm, and explainability information via an interactive GUI. The execution testing in Chapter 5.3.1 revealed that the target group is enabled to interact with Meta-DPP, since no errors occurred during execution, while Meta-DPP always provided an outcome. In conclusion, the requirement is deemed *completely fulfilled*.

The *response latency* needed to be within 5 minutes to account for a comprehensive computing within a reasonable amount of time. When applying Meta-DPP on the 17 data sets used in Chapter 5, the response time was 3.5 minutes on average meaning that the requirement is *entirely fulfilled*. Besides the response latency, Meta-DPP provides *ranked recommendations*. The display of probabilities does not only indicate the certainty of the meta model, but also serves as a ranking as shown when comparing two production use cases in each scenario in Chapter 5.2. Therefore, the requirement of ranked recommendations is *entirely fulfilled*.

In case the user does not provide every detail of the production use case, Meta-DPP still gives a recommendation as checked in the special input testing in Chapter 5.3.1. If information about ML algorithms or explainability is missing, Meta-DPP proved the *input flexibility* as discussed in Chapter 5.3. In addition, Meta-DPP provides warning

messages in case non-conform data sets are inserted or the wrong learning task is chosen based on the provided data set leading to an *entire fulfillment* of this requirement.

The *explainability* requirement is met by filtering the DPP pipelines from the pipeline pool according to the input from the user. In case the user checks the explainability requirement, only DPP pipelines from the pipeline pool are displayed, which contain transparent DPP methods. In addition, the meta models selected in Chapter 4.4.1 are explainable to comprehend the decisions of Meta-DPP. Since the explainability information is handled by Meta-DPP, the requirement is *completely fulfilled*. Figure 5-28 summarizes the results of this requirements review. 14 out of 16 requirements are an *completely fulfilled*.

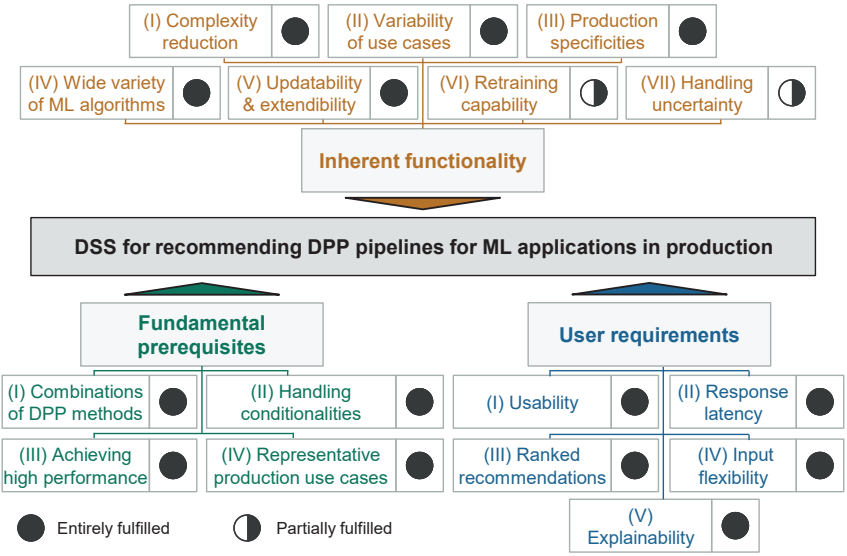


Figure 5-28: Results of the requirements assessment

5.4 Interim conclusion

Chapter 5 aimed at validating Meta-DPP based on a case study consisting of 324 production use cases. In addition, verification was performed by performing both execution and special input testing on the implemented Meta-DPP, while subsequently, the fulfillment of requirements was assessed. The case study revealed that the developed benchmarking and pipeline pool outperforms essential pipelines in all five scenarios on average. In addition, the meta model showed the potential in recommending DPP pipelines from the pool when focusing on the achieved DPP scores from recommended to essential pipelines. The execution and special input testing showed that Meta-DPP is robust to input uncertainty and navigates the user in case of wrong inputs. Ultimately, the main research question can be answered:

MRQ Can a decision support system be developed that supports in recommending DPP pipelines for ML applications in production?

Based on the results of verification and validation, the positive assessment of the requirements and by taking into account the answered SRQs in Chapter 3 and 4, the *MRQ is positively answered*. The different testing techniques reveal that Meta-DPP runs stable and enables the user in applying DPP pipelines for a given production use case that focuses on supervised learning algorithms and tabular data. The case study for validating Meta-DPP proved the overall performance and competitiveness against the defined baselines. 14 from 16 requirements are entirely fulfilled, while two requirements are partially fulfilled.

6 Critical Discussion

Based on the developed, verified and validated Meta-DPP, Chapter 6 critically discusses the methodological approach, design decisions, and implementations of Meta-DPP. As shown in the verification and validation, Meta-DPP is capable of recommending performant DPP pipelines for ML applications in production. However, every design decision performed for developing Meta-DPP is ambivalent meaning that a choice for an approach or design simultaneously results in a deliberate exclusion of another opportunity. Thus, all design decisions are critically discussed following the component structure of Meta-DPP, which are the meta target selector, meta features database and meta model and their individual sub components. Finally, verification and validation are criticized and final remarks are provided regarding the impact of DPP on ML model performances in production.

Meta target selector

Within the *creation of production use cases*, 104 data sets have been created serving as representative data basis. Even if it could be disproved that the scores are different across the derivatives within the benchmarking and different pipelines belong to the best performers, the creation of derivatives may introduce noise in the meta data set since they are only slightly modified. Thereby, the data sets become too laboratory, while not representing real production environments. Adding derivative data sets can then lead to worse generalization ability of the meta model. In the future, the meta model can be trained without derivatives. Even though the production use cases are representative, data sets with a very high number of instances and features are rather rare. Further works can also concentrate on more extreme cases regarding the number of missing values, categorical features, or outliers.

The *configuration of suitable DPP pipelines* is built on a thorough assessment to identify DPP methods and subsequently combine these methods into meaningful pipelines. Compared to the existing approaches shown in Chapter 3.2, DPP methods from several steps and categories are thereby considered. DPP methods were selected if the method exhibits the highest assessment score. In the future, a threshold can also be chosen as trade off to have enough heterogeneous methods but also to keep the number of DPP methods low in total. This threshold can be adjusted in the future to get other DPP methods. The assessment of DPP methods can further be extended through cost utility analyses towards the selection of more performant yet less explainable methods. For instance, instead of using *MeanImputer* or *MedianImputer* for missing value handling, *K-NN* could be used for imputation.

When configuring the DPP methods into pipelines, the large configuration space was addressed by keeping the order of DPP methods within the pipelines fixed. In addition, a DPP method was only selected once per DPP category. These decisions narrow down the number of combinations at the expense of better pipeline performances. For instance, categorical columns in production use cases may be nominal and ordinal requiring two different types of encoding, which is not handled by the benchmarking in its current form. Lastly and according to Figure 1-1, data preparation comprises DPP and feature engineering. Therefore, further methods from feature engineering can be included in the preparation of data.

The *benchmarking* of DPP pipelines over 1,872 production use cases considers different pipeline lengths, conditionalities, and a train test split prior to preprocessing the data. An advantage of the benchmarking is the opportunity of generating fully comprehensive ML pipelines independent from DPP. A user could also run the benchmarking for one ML algorithm of choice instead of running all 18 ML algorithms at once. The train test split was performed based on a ratio of 70/30 to account for training times. In the future, different train test splits, e. g., 80/20 can be realized. The benchmarking can further be extended by taking into account additional ML algorithms, such as deep learning algorithms or by inserting consistency checks as provided by *Potter's Wheel* and *HoloClean* (see Chapter 3.2). As shown in the benchmarking results in Chapter 4.2.3, many DPP pipelines show a poor performance. Therefore, the number of pipelines can be reduced to speed up the benchmarking. One approach is to sort the median DPP scores in an ascending manner and identify the worst performing pipelines. DPP methods can then be removed based on the occurrences of DPP methods in the worst performing pipelines. The design of putting encoded features aside is technically meaningful, however, when a *OneHotEncoder* is used in columns with high cardinality, the number of features grow exponentially forcing the curse of dimensionality. These features are not treated within the feature selection due to implementation reasons. Future work can focus on bringing back the encoded features before feature selection to manage the number of columns.

The *scaled ranking* provides comparable DPP scores over different use cases for both classification and regression. DPP scores can then be used to identify highly performant DPP pipelines. Chapter 4.2.5 showed that DPP scores for regression were significantly higher than for classification. This can stem from various reasons such as the selected data sets or naturally because of the learning task. However, compared to the F1 score, NRMSE represents an unbounded measure, while the benchmarking showed that outliers are available in rare cases. These outliers were not removed but deliberately kept within the ranking to not affect the actual distribution. However, due

to the findings of the benchmarking and since the scaled ranking as applied in Chapter 4.2.4 is susceptible to outliers, many high DPP scores are more likely available. Future works may focus on bounding the NRMSE value (e. g., to NRMSE = 2) to remove the outliers or on testing the performance of a robust scaler introduced in Chapter 2.4.3.

Pooling determines the best performing pipeline based on the assumption of a dependency between ML algorithms and DPP pipelines. The pooling procedure also accounts for the variable number of pipelines in the pipeline pool, which represents a critical hyperparameter of Meta-DPP. In Chapter 4.2.5, the number was set to four pipelines which addresses the meta target tradeoff. Different number of pipelines were tested during the development, in which four pipelines represented the best tradeoff. If a user is interested in high upper bounds and wants to test several pipelines instead of focusing on a highly performant meta model, the user can select up to 304 DPP pipelines for regression and 608 for classification. Due to the high dependency of ML algorithms and DPP, another approach is to split the pipeline pool according to ML algorithms leading to 36 pipeline pools for classification and regression. This approach was deemed inappropriate in this thesis due to the low number of regression use cases, since the meta model would have been trained on only 29 instances. However, as the number of data sets increases over time, this approach may be meaningful. Besides that, the pooling of pipelines was performed based on the benchmarking results of the same data sets, which can foster data leakage. This decision was made due to the limited amount of production use cases and is assessed as not being an issue as validation results showed similar pooling results.

Meta features database

The meta features database consists of data set-, ML algorithm-, and DPP pipeline-related meta features. 188 meta features were determined for classification and 189 for regression identified in a brute force approach. Even though the meta features can bring more information, the curse of dimensionality remains an issue. This can be seen in the feature importances and also in the final meta models, which do not consider pipeline-specific meta features and in case of classification, only applies the 30 most important features. For the *extra trees classifier*, 27 out of 30 most important features are related to ML algorithm performance. The *gradient boosting classifier* uses 16 data set- and 14 ML algorithm-related meta features. All groups of *data set related meta features* are represented within the important features, which forces to introduce further groups. According to *pymfe*, further groups are related to clustering, complexity, or concept [ALCO20]. For the different statistical, information-theoretical, and landmarking, the mean and standard deviations are used for retrieving information of, e. g., the error rates of a worst node (see Figure 4-23). However, the mean and standard

deviation assumes a normal distribution, which in real use cases is not available as shown in Figure 4-23. Future works can focus on calculating different statistical location measures, which are suitable depending on the underlying distribution.

Since *ML algorithm-related meta features* highly influence the choice of ML algorithms, further meta features can be derived from the benchmarking. For instance, instead of using the average performance of ML algorithms, different statistical location measures can be used to provide information about the underlying task. *DPP pipeline-related meta features* are not used by the meta models. However, further benchmarking results can be incorporated and tested. Three top performing pipelines have been chosen per ML algorithm. This hyperparameter can be adjusted to count the occurrences of DPP methods within the pipelines.

Meta model

The *selection* of meta pipelines resulted in the choice of baseline models as well as commonly used ML algorithms in meta learning. To further increase the performance of the meta model, the benchmarking can be applied to identify best performing meta pipelines. Instead of using production use cases, the final meta data set can be used for benchmarking ML pipelines. The top performing ML pipeline for the regression and classification meta data set can then be reimplemented and used as meta pipeline.

Within *training* the meta pipelines, a design of experiments was introduced, which covers different ML algorithms, different sets and numbers of meta features. To address the curse of dimensionality, the number of meta features has been narrowed down given the feature importances. Since the feature importance values were not high for both regression and classification, 20 and 30 features were chosen to be included. However, different settings can be tested to achieve higher DPP scores and model performances. In addition, the optimization of ML algorithm's hyperparameter and the choice of systems based on AutoML or learning to rank can increase the performance of the meta model.

Instead of training a meta model, a rule based approach could have been chosen to display, for instance, the top 10 pipelines dependent on the ML algorithm. Since the meta model shows promising results and learns over time when more use cases are incorporated, a meta learning approach was deliberately chosen over a simpler rule based approach. However, when the pooling is made dependent on the ML algorithm, a rule based approach may lead to higher DPP scores.

Verification and validation

The results of the verification and validation show that Meta-DPP is capable of recommending DPP pipelines for ML applications in production. Top performing pipelines

from benchmarking, pipeline pool, and pipelines recommended by the meta model generally outperform essential pipelines for both classification and regression on average across 324 production use cases. While for classification, the recommendations of the meta model are distributed over several classes, the meta model for regression (*gradient boosting*) recommends the DPP pipeline ID00005 in 85 % of all cases. High accuracy scores are achieved since the data sets are also imbalanced towards this pipeline. The actual distribution of best performing pipelines show that one pipeline outperforms the other pipelines in 50 % of all cases. Since the benchmarking only reveals the actual pipeline performances, the performance distribution could not have known beforehand to ensure a balanced distribution of actually best performing pipelines from the pool. When assuming that the data sets are representative production use cases, future work may focus on the development of another pooling strategy to determine different yet more heterogeneous pipelines. However, since the meta model shows a high imbalance in the recommendation, the model and underlying meta data set needs to be adjusted. Within the validation and occurrence of wrong predictions by the meta model, it was observed that the probability of the meta model outputs decreases in recommending the pipeline ID 00005. This may be an indicator for the necessity of further data points since the meta model seems to identify a pattern when being less certain in choosing the overrepresented class. Compared to classification, the number of data points within training was only 442 compared to 1,260 classification data sets. Since the meta data set for regression showed a slight imbalance, future work may also concentrate on performing balancing through over or undersampling prior to model training or introduce new data sets and ML algorithms for the further development of Meta-DPP.

For classification, machine tool degradation derivatives lead to similar results, while for regression, essential pipelines outperformed the meta model and pipeline pool for the derivative *both*. Thus, the choice of production use cases is deemed appropriate. However, future work needs to focus on considering additional use cases also from other ML applications (see Chapter 2.1)

Final remarks on the impact of DPP on ML model performances in production

The development and application of Meta-DPP on production use cases has revealed and proven the high impact of DPP on ML model performances, which is shown by the full bandwidth of performances regarding F1 scores [0,1] and NRMSE values [0,2) through the benchmarking (Chapter 4.2.3). Further referring to the production domain, versatile kinds of quality issues are present in raw tabular data making DPP especially important and complex. The different DPP steps covered in this work have proven to address the quality issues such as outliers, highly imbalanced data, or missing values

(see Chapter 2.1 and 5.1). Meta-DPP contributes to the increase of data quality by providing DPP pipelines that show error-resistant behavior yet being performant.

7 Summary and Outlook

7.1 Summary

The era of Industry 4.0 opens up the possibility of optimizing production systems in a data-driven way. To turn data into value, machine learning (ML) models are trained on production data aiming at identifying patterns to optimize processes in the tension triangle of time, quality and cost. A crucial prerequisite for achieving performant ML models is the availability of high quality data. Since raw data generated in production exhibits very different kinds of quality issues, data preprocessing (DPP) is required to increase the quality of the data. One of the key design decisions in any ML project is the choice of suitable DPP methods. These methods can be chosen within the frame of DPP steps, which are: data cleaning, transformation, reduction, as well as augmentation and balancing. Due to the high number of possible DPP methods, data scientists commonly select suitable DPP methods manually and via trial and error. The search space further increases significantly when DPP methods are configured into DPP pipelines. For these reasons, DPP nowadays accounts for approximately 80 % of the time in ML projects.

To guide data scientists, decision support systems have been developed that assist in the selection of suitable DPP pipelines. The existing approaches build on promising technologies such as meta learning or automated machine learning (AutoML) but do not cover production-specific requirements as well as methods from every DPP step. Therefore, the main research question was derived: Can a decision support system be developed that supports in recommending DPP pipelines for ML applications in production?

To be able to answer the main research question, a meta learning-based decision support system, called Meta-DPP, was developed. Meta-DPP relies on three core components: the meta target selector, meta features database, and meta model. The *meta target selector* chooses between two preselected sets of overall well performing pipelines, called pipeline pools, for both classification and regression tasks. This component was developed in five steps. First, 1,872 production use cases were created based on the combination of data sets and ML algorithms. Subsequently, DPP pipelines were configured out of 21 preselected DPP methods. For every production use case, DPP pipelines were benchmarked. The performance of DPP pipelines were determined based on the performance of ML algorithms. To identify overall best performing pipelines, a scaled ranking was applied. For managing the number of pipelines the meta model needs to classify, a pooling was performed that takes into account the high

dependence between ML algorithm and DPP pipelines. Ultimately, the meta target selector chooses between two pipeline pools of either regression or classification.

Afterwards, the *meta features database* was set up, which consists of data set-, ML algorithm-, and DPP pipeline-related meta features. While data set-related features such as statistical or information-theoretical meta features are calculated based on the data set at hand, performances of ML algorithms and DPP pipelines were calculated from the benchmarking and scaled ranking results. Finally, the *meta model* was developed by identifying suitable meta pipelines. The selected meta pipelines for the two pipeline pools were trained on the meta data sets stemming from the 1,872 production use cases. To eventually receive a recommendation, Meta-DPP was implemented. A user interface enables the data scientist, production expert, or IT expert to input their data set, learning task, ML algorithm and information about explainability. Given the input, Meta-DPP provides a ranked recommendation of the DPP pipelines from the pool. The probabilities further indicate how certain Meta-DPP is about the recommendation.

Verifying and validating revealed the correct development and implementation of Meta-DPP. The validation on 324 production use cases further prove that Meta-DPP outperform essential pipelines on average, which only realize essential DPP for the functioning of ML algorithms. As a conclusion, the main research question was positively answered.

7.2 Outlook

The presented results offer the opportunity for future research that can be classified into six aspects. The aspects are ordered by the expected time line starting with short term and ending up to long term.

(1) Applying Meta-DPP on more production use cases & further applications

Meta-DPP was validated on 324 production use cases stemming from three applications: predictive maintenance, predictive process control, and process design. Given the prerequisites of the availability of a supervised learning problem and tabular data set, Meta-DPP can be applied to more production use cases in the field of these three applications. In addition, Meta-DPP can be transferred to further production applications such as anomaly detection, or process management.

(2) Optimizing Meta-DPP considering core components & retraining concept

Three components build the core of Meta-DPP. Starting with the *meta target selector*, future work may focus on considering hyperparameter tuning of DPP methods to increase the overall benchmarking performance. The *scaled ranking* approach can be

further optimized by bounding RMSE values to be less prone to outliers. In terms of *pooling*, the hyperparameter of the number of pipelines, which belong to a pool, can be adjusted in the future. New pooling approaches, which entirely focus on algorithm-specific DPP pipelines can be followed. The *meta features database* can be extended by introducing further results from the benchmarking or by extracting further meta features from the data set. Since the *meta model* consists of a meta pipeline of an encoder and a meta algorithm, the benchmarking could be used to identify best performing meta pipelines. Besides the optimization of core components, the retraining concept can be implemented.

(3) Extending the components of Meta-DPP

In the future, DPP methods can be added or substituted to increase the overall performance. Meta-DPP could provide the possibility of incorporating methods of feature engineering to depict domain specific knowledge. To increase the number of production use cases and to enhance the performance of Meta-DPP, further ML algorithms can be introduced in the future. Meta-DPP can also be extended by considering other learning approaches, e. g., unsupervised learning with a clustering task, or new data types like image data, video, or audio. The concept of Meta-DPP based on meta target selector, meta features database and meta model can then be transferred to these data types. Future work can also focus on the explainability of Meta-DPP and give information on how Meta-DPP comes to a specific recommendation.

(4) Transferring the concept of Meta-DPP to other steps of the ML pipeline

Since Meta-DPP builds on meta learning, the concept can be transferred to other design decisions being made within the ML pipeline, namely the choice of data integration techniques, the selection of ML algorithms as well as hyperparameter optimization methods, and the choice of the deployment strategy. In addition, as the concept of meta learning is widely applicable, Meta-DPP can support in the selection of uncertainty quantification or explainable AI methods.

(5) Combining Meta-DPP with other developments along the ML pipeline

Besides the transfer of Meta-DPP to other design decisions, Meta-DPP can be combined with other developments within the ML pipeline. For instance, a belief rule based expert system (BRBES) was developed for recommending hyperparameter optimization techniques in production [KRAU22]. While Meta-DPP provides a DPP pipeline, the BRBES recommends a hyperparameter optimization technique for a given ML algorithm. Future work can focus on combining these two approaches to recommend DPP pipelines and hyperparameter optimization techniques based on an ML algorithm to optimize the ML pipeline more comprehensively.

(6) Applying components of Meta-DPP for an early identification of the potential of an ML use case

The benchmarking as one component of Meta-DPP can be used to identify the potential of an ML use case. Once data is available from an application, the benchmarking routine can be started. The achieved F1 scores and RMSE values then indicate the general performance and can give a decision basis for choosing suitable ML use cases in production. Future work could focus on quantifying, under which circumstances the benchmarking provides a profound decision basis to identify the potential of ML use cases.

V Bibliography

[ABDA17] Abdallah, Z. S.; Du, L.; Webb, G. I.: Data Preparation, 2017, pp. 318–327, DOI: 10.1007/978-1-4899-7687-1_62.

[ACHL01] Achlioptas, D.: Database-friendly random projections. In: Proceedings of the twentieth ACM, 2001, pp. 274–281, DOI: 10.1145/375551.375608.

[AGOG07] Agogino A.; Goebel K.: Milling Data Set. URL: http://ti.arc.nasa.gov/project/prognostic-data-repository [Accessed: 03-Feb-22]

[AKRI19] Akrichi, S.; Abbassi, A.; Abid, S.; Noureddine, B. y.: Roundness and positioning deviation prediction in single point incremental forming using deep learning approaches. In: Advances in Mechanical Engineering, 2019, DOI: 10.1177/1687814019864465.

[ALCO20] Alcobaça, E.; Siqueira, F.; Rivolli, A.; Garcia, L. P. F.; Oliva, J. T.; Carvalho, A. C. P. L. F. de: MFE: Towards reproducible meta-feature extraction. In: Journal of Machine Learning Research, 2020, No. 21(111), pp. 1–5

[ALPA14] Alpaydin, E.: Introduction to machine learning Series: Adaptive computation and machine learning series. Third edition ed. Cambridge, Massachusetts: MIT Press, 2014, ISBN: 9780262325745

[AMAZ20] Amazon Web Services, Inc.: Amazon SageMaker - Developer Guide. URL: https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-dg.pdf [Accessed: 08.09.2021]

[ANGI07] Angiulli, F.: Fast Nearest Neighbor Condensation for Large Data Sets Classification. In: IEEE Transactions on Knowledge and Data Engineering, Vol. 19, 2007, No. 11, pp. 1450–1464, DOI: 10.1109/TKDE.2007.190645.

[APEL15] Apel, D.; Behme, W.; Eberlein, R.; Merighi, C.: Datenqualität erfolgreich steuern. Praxislösungen für Business-Intelligence-Projekte Series: Safari Tech Books Online. 3rd ed. Heidelberg: dpunkt.verl., 2015, ISBN: 3864900425

[ARNO14] Arnott, D.; Pervan, G.: A Critical Analysis of Decision Support Systems Research Revisited: The Rise of Design Science. In: Journal of Information Technology, 2014, DOI: 10.1057/jit.2014.16.

[ARON20] Aronio de Romblay, A.: MLBox Documentation. URL: https://mlbox.readthedocs.io/en/latest/index.html [Accessed: 08.09.2021]

[AWAD15] Awad, M.; Khanna, R.: Efficient learning machines. Theories, concepts, and applications for engineers and system Designers Series:

The expert's voice in machine learning. New York: Apress Open, 2015, ISBN: 1430259906

[BADR19] Badr, W.: 6 Different Ways to Compensate for Missing Values In a Dataset (Data Imputation with examples). URL: https://towardsdatascience.com/6-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779 [Accessed: 16.08.2021]

[BALC98] Balci, O.: Verification, Validation, and Testing. In: Handbook of simulation: principles, methodology, advances, applications, and practice, 1998

[BARA01] Barandela, R., N. Cortés, and A. Palacios: The nearest neighbor rule and the reduction of the training sample size. In: Proceedings 9th Symposium on Pattern Recognition and Image Analysis, Vol. 1, 2001

[BARN99] Barnard, J.; Meng, X.-L.: Applications of multiple imputation in medical studies: from AIDS to NHANES. In: Statistical Methods in Medical Research, Vol. 8, 1999, No. 1, pp. 17–36, DOI: 10.1191/096228099666230705.

[BATI04] Batista, G. E. A. P. A.; Prati, R. C.; Monard, M. C.: A study of the behavior of several methods for balancing machine learning training data. In: ACM SIGKDD Explorations Newsletter, Vol. 6, 2004, No. 1, pp. 20–29, DOI: 10.1145/1007730.1007735.

[BAUD00] Baudat, G.; Anouar, F.: Generalized discriminant analysis using a kernel approach. In: Neural Computation, Vol. 12, 2000, No. 10, pp. 2385–2404, DOI: 10.1162/089976600300014980.

[BEKK13] Bekkar, M.; Djemaa, H. K.; Alitouche, T. A.: Evaluation Measures for Models Assessment over Imbalanced Data Sets. In: Journal of Information Engineering and Applications, Vol. 3, 2013, No. 10

[BELL21] Belle, V.; Papantonis, I.: Principles and Practice of Explainable Machine Learning. In: Frontiers in Big Data, Vol. 4, 2021, DOI: 10.3389/fdata.2021.688969.

[BELL61] Bellman, R. E.: Adaptive Control Processes Series: Princeton Legacy Library, 2045, 1961, DOI: 10.1515/9781400874668.

[BENS00a] Bensusan, H.; Giraud-Carrier, C.; Kennedy, Claire, J.: A Higher-order Approach to Meta-learning. In: Inductive Logic Programming, 10th International Conference, ILP 2000, Work-in-progress reports, London, UK, 2000

[BENS00b] Bensusan, H.; Giraud-Carrier, C.: Discovering Task Neighbourhoods Through Landmark Learning Performances. In: Zighed, D.A., Komorowski, J., Żytkow, J. (eds) Principles of Data Mining and Knowledge

Discovery. PKDD 2000. Lecture Notes in Computer Science, Vol. 1910, 2000, DOI: 10.1007/3-540-45372-5_32.

[BERG12] Bergstra, J.; Bengio, Y.: Random Search for Hyper-Parameter Optimization. In: Journal of Machine Learning Research, Vol. 13, 2012, pp. 281–305

[BERT09] Berthold, M. R.; Cebron, N.; Dill, F.; Gabriel, T. R.; Kötter, T.; Meinl, T.; Ohl, P.; Thiel, K.; Wiswedel, B.: Knime - the konstanz information miner: version 2.0 and beyond. In: ACM SIGKDD Explorations Newsletter, Vol. 11, 2009, No. 1, DOI: 10.1145/1656274.1656280.

[BERT19] Berti-Equille, L.: Learn2Clean: Optimizing the Sequence of Tasks for Web Data Preparation, 2019, pp. 2580–2586, DOI: 10.1145/3308558.3313602.

[BERT21] Berti-Equille, L.; Comignani, U.: Explaining Automated Data Cleaning with CLeanEX. In: IJCAIPRICAI-2020 Workshop on Explainable Artificial Intelligence (XAI), 2021

[BEZD81] Bezdek, J. C.: Pattern Recognition with Fuzzy Objective Function Algorithms Series: Advanced Applications in Pattern Recognition Ser. New York, NY: Springer, 1981, ISBN: 9781475704501

[BHAA17] Bhattacharya, G.; Ghosh, K.; Chowdhury, A. S.: kNN Classification with an Outlier Informative Distance Measure, In: Shankar, B., Ghosh, K., Mandal, D., Ray, S., Zhang, D., Pal, S. (eds) Pattern Recognition and Machine Intelligence. PReMI 2017. Lecture Notes in Computer Science: Springer, Cham, vol. 10597, 2017, DOI: 10.1007/978-3-319-69900-4_3.

[BHAN20] Bhandari, A.: Feature Scaling for Machine Learning: Understanding the Difference Between Normalization vs. Standardization. URL: https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/ [Accessed: 27.07.2021]

[BILA18] Bilalli, B.; Abelló, A.; Aluja-Banet, T.; Wrembel, R.: PRESISTANT: Learning based assistant for data pre-processing, 2018, DOI: 10.48550/arXiv.1803.01024.

[BIRG17] Birgelen, A. von; Niggemann, O.: Using Self-Organizing Maps to Learn Hybrid Timed Automata in Absence of Discrete Events. In: 22nd IEEE International Conference on Emerging Technologies and Factory Automation, 2017

[BIRG18a] Birgelen, A. von; Buratti, D.; Mager, J.; Niggemann, O.: Self-Organizing Maps for Anomaly Localization and Predictive Maintenance in Cyber-Physical Production Systems. In: 51st CIRP Conference on

Manufacturing Systems (CIRP CMS 2018), 2018, No. 72, pp. 480–485, DOI: 10.1016/j.procir.2018.03.150.

[BIRG18b] Birgelen, A. von; Niggemann, O.: Anomaly Detection and Localization for Cyber-Physical Production Systems with Self-Organizing Maps, In: Niggemann, O., Schüller, P. (eds) IMPROVE - Innovative Modelling Approaches for Production Systems to Raise Validatable Efficiency. Technologien für die intelligente Automation. In: Springer Vieweg, Berlin, Heidelberg, Vol. 8, 2018, 55–71, DOI: 10.1007/978-3-662-57805-6_4.

[BIRG18c] Birgelen von, A.; Niggemann, O.: Enable learning of Hybrid Timed Automata in Absence of Discrete Events through Self-Organizing Maps, In: Niggemann, O., Schüller, P. (eds) IMPROVE - Innovative Modelling Approaches for Production Systems to Raise Validatable Efficiency. Technologien für die intelligente Automation. In: Springer Vieweg, Berlin, Heidelberg, Vol. 8, 2018, pp. 37–54, DOI: 10.1007/978-3-662-57805-6_3.

[BISH06] Bishop, C. M.: Pattern recognition and machine learning Series: Information science and statistics. New York, NY: Springer, 2006, ISBN: 0387310738

[BISH98] Bishop, C. M.: Bayesian PCA. In: Advances in Neural Information Processing Systems, Vol. 11, 1998, pp. 509–514

[BITK20] Bitkom e.V.: Industrie 4.0 – so digital sind Deutschlands Fabriken. URL: https://www.bitkom.org/Presse/Presseinformation/Industrie-40-so-digital-sind-Deutschlands-Fabriken [Accessed: 02.08.2021]

[BLAN13] Blanca, M. J.; Arnau, J.; López-Montiel, D.; Bono, R.; Bendayan, R.: Skewness and kurtosis in real data samples. In: Methodology, 2013, No. 9, pp. 78–84, DOI: 10.1027/1614-2241/a000057.

[BOSE92] Boser, B. E.; Guyon, I. M.; Vapnik, V. N.: A training algorithm for optimal margin classifiers, In: COLT '92: Proceedings of the fifth annual workshop on Computational learning theory, 1992, pp. 144–152, DOI: 10.1145/130385.130401.

[BOWN16] Bowne-Anderson, H.: Preprocessing in Data Science (Part 2). URL: https://www.datacamp.com/community/tutorials/preprocessing-in-data-science-part-2-centering-scaling-and-logistic-regression [Accessed: 29.07.2021]

[BOX64] Box, G. E. P.; Cox, D. R.: An Analysis of Transformations. In: Journal of the Royal Statistical Society: Series B (Methodological), Vol. 26, 1964, No. 2, pp. 211–243, DOI: 10.1111/j.2517-6161.1964.tb00553.x.

[BRAN02] Brand, M.: Incremental Singular Value Decomposition of Uncertain Data with Missing Values. In: Heyden, A., Sparr, G., Nielsen, M., Johansen, P. (eds) Computer Vision — ECCV 2002. Lecture Notes in Computer ScienceEuropean Conference on Computer Vision, Vol. 2350, 2002, pp. 707–720, DOI: 10.1007/3-540-47969-4_47.

[BRAN17] Branco, P., Torgo, L., Ribeiro, R.: SMOGN: a Pre-processing Approach for Imbalanced Regression. In: Proceedings of Machine Learning Research, Vol. 74, 2017, pp. 36–50

[BRAN19] Brandstätter, T. C.; Krauß, J.; Schmitt, R. H.: Certification of AI-Supported Production Processes. In: Wulfsberg, J.P. ; Wissenschaftliche Gesellschaft für Produktionstechnik -WGP, 2019, pp. 553–561, DOI: 10.1007/978-3-662-60417-5_55.

[BRAZ09] Brazdil, P.; Gabbay, D. M.; Giraud-Carrier, C.; Siekmann, J.; Soares, C.; Vilalta, R.: Metalearning. Applications to Data Mining Series: Cognitive Technologies. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, ISBN: 9783540732631, DOI: 10.1007/978-3-540-73263-1.

[BREI01] Breiman, L.: Random Forests. In: Machine Learning, Vol. 45, 2001, No. 1, pp. 5–32, DOI: 10.1023/A:1010933404324.

[BREI84] Breiman, L.; Friedman, J.; Stone, C. J.; Olshen R.A.: Classification and Regression Trees. Wadsworth: Belmont, CA, 1984, DOI: 10.1201/9781315139470.

[BREU00] Breunig, M. M.; Kriegel, H.-P.; Ng, R. T.; Sander, J.: LOF. In: ACM SIGMOD Record, Vol. 29, 2000, No. 2, pp. 93–104, DOI: 10.1145/335191.335388.

[BROC02] Brockwell, P.; Davis, R.: Introduction to time series and forecasting. 2 ed. New York, Berlin, Heidelberg: Springer, 2002, ISBN: 0-387-95351-5

[BROW18] Brownlee, J.: Train Neural Networks With Noise to Reduce Overfitting. URL: https://machinelearningmastery.com/train-neural-networks-with-noise-to-reduce-overfitting/ [Accessed: 28.07.2021]

[BROW19a] Brownlee, J.: 3 Ways to Encode Categorical Variables for Deep Learning. URL: https://machinelearningmastery.com/how-to-prepare-categorical-data-for-deep-learning-in-python/ [Accessed: 28.07.2021]

[BROW19b] Brownlee, J.: How to use Data Scaling Improve Deep Learning Model Stability and Performance. URL: https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/ [Accessed: 29.07.2021]

[BROW19c] Brownlee, J.: How to Choose a Feature Selection Method For Machine Learning. URL: https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/ [Accessed: 08.08.2021]

[BROW20a] Brownlee, J.: Boosting and AdaBoost for Machine Learning. URL: https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/ [Accessed: 08.08.2021]

[BROW20b] Brownlee, J.: How to Perform Data Cleaning for Machine Learning with Python. URL: https://machinelearningmastery.com/basic-data-cleaning-for-machine-learning/ [Accessed: 27.07.2021]

[BROW20c] Brownlee, J.: Data Preparation for Machine Learning. Data Cleaning, Feature Selection and Data Transforms in Python, 2020

[BROW20d] Brownlee, J.: Cost-Sensitive Logistic Regression for Imbalanced Classification. URL: https://machinelearningmastery.com/cost-sensitive-logistic-regression/ [Accessed: 29.07.2021]

[BROW20e] Brownlee, J.: How to Use Power Transforms for Machine Learning. URL: https://machinelearningmastery.com/power-transforms-with-scikit-learn/ [Accessed: 27.07.2021]

[BUIH20] Bui, H.: How to Encode Categorical Data - Towards Data Science. URL: https://towardsdatascience.com/how-to-encode-categorical-data-d44dde313131 [Accessed: 07.08.2021]

[BUND21] Bundesministerium für Wirtschaft und Energie: Digitale Transformation in der Industrie. URL: https://www.bmwi.de/Redaktion/DE/Dossier/industrie-40.html [Accessed: 02.08.2021]

[BURD20] Burdack, J.; Horst, F.; Giesselbach, S.; Hassan, I.; Daffner, S.; Schöllhorn, W. I.: Systematic Comparison of the Influence of Different Data Preprocessing Methods on the Performance of Gait Classifications Using Machine Learning. In: Frontiers in bioengineering and biotechnology, Vol. 8, 2020, p.260, DOI: 10.3389/fbioe.2020.00260.

[BUTE20] Buteikis, A.: Practical Econometrics and Data Science. Vilnius University: Faculty of Mathematics and Informatics, Institute of Applied Mathematics. URL: http://web.vu.lt/mif/a.buteikis/wp-content/uploads/PE_Book/index.html

[BUUR11] van Buuren, S.; Groothuis-Oudshoorn, K.: mice: Multivariate Imputation by Chained Equations in R. In: Journal of Statistical Software, Vol. 45, 2011, No. 3, DOI: 10.18637/jss.v045.i03.

[CAMP13] Campello, Ricardo J. G. B.; Moulavi, D.; Sander, J.: Density-Based Clustering Based on Hierarchical Density Estimates. In: Pei, J., Tseng, V.S., Cao, L., Motoda, H., Xu, G. (eds) Advances in Knowledge Discovery and Data Mining. PAKDD 2013. Lecture Notes

in Computer Science, Vol. 7819, 2013, No. Springer, Berlin, Heidelberg, pp. 160–172, DOI: 10.1007/978-3-642-37456-2_14.

[CARE03] Carey, G.: Coding Categorical Variables. Boulder, Colorado, USA, 2003

[CERQ97] Cerquides; Jesús; López de Màntaras; Ramon: Proposal and Empirical Comparison of a Parallelizable Distance-Based Discretization Method, January 1997

[CHAK19] Chakure, A.: Support Vector Machines (SVMs) - DataDrivenInvestor. URL: https://medium.datadriveninvestor.com/support-vector-machines-svms-4bcccbd78369 [Accessed: 29.07.2021]

[CHAN03] Chan, Y.: Biostatistics 104: correlational analysis. In: Singapore medical journal, Vol. 44, 2003, No. 12, pp. 614–619

[CHAO05] Chao, S.; Li, Y.: Multivariate Interdependent Discretization for Continuous Attribute. In: Procedia, 2005, pp. 167–172, DOI: 10.1109/ICITA.2005.188.

[CHAP00] Chapman, P.; Clinton, J.; Kerber, R.; Khabaza, T.; Reinartz, T.; Shearer, C.; Wirth, R.: CRISP-DM 1.0. Step-by-step data mining guide. URL: https://the-modeling-agency.com/crisp-dm.pdf [Accessed: 08.09.2021]

[CHAW02] Chawla, N. V.; Bowyer, K. W.; Hall, L. O.; Kegelmeyer, W. P.: SMOTE: Synthetic Minority Over-sampling Technique. In: Journal of Artificial Intelligence Research, Vol. 16, 2002, pp. 321–357, DOI: 10.1613/jair.953.

[CHEN16] Chen, T.; Guestrin, C.: XGBoost: A Scalable Tree Boosting System. In: KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Vol. 11, 2016, No. 34, pp. 785–794, DOI: 10.1145/2939672.2939785.

[CHEN17] Chen, Y.; Sun, R.; Gao, Y.; Leopold, J.: A nested-ANN prediction model for surface roughness considering the effects of cutting forces and tool vibrations. In: Measurement, 2017, No. 98, pp. 25–34, DOI: 10.1016/j.measurement.2016.11.027.

[CHEN21] Chen, L.: From imbalanced datasets to boosting algorithms. Imbalanced dataset full tool kit. URL: https://towardsdatascience.com/from-imbalanced-dataset-to-boosting-algorithms-1-2-798cd6384ecc [Accessed: 22.07.2021]

[CHIN95] Ching, J. Y.; Wong, A.; Chan, K.: Class-dependent discretization for inductive learning from continuous and mixed-mode data. In: IEEE transactions on pattern analysis and machine intelligence, Vol. 17, 1995, No. 7, pp. 641–651, DOI: 10.1109/34.391407.

[CHIU17] Chiu, H.-W.; Lee, C.-H.: Prediction of machining accuracy and surface quality for CNC machine tools using data driven approach. In: Advances in Engineering Software, 2017, No. 114, DOI: 10.1016/j.advengsoft.2017.07.008.

[CHON20] Chong, J.: What is Feature Scaling & Why is it Important in Machine Learning? URL: https://towardsdatascience.com/what-is-feature-scaling-why-is-it-important-in-machine-learning-2854ae877048 [Accessed: 07.08.2021]

[CHOU06] Chou, C.-H.; Kuo, B.-H.; Chang, F.: The Generalized Condensed Nearest Neighbor Rule as A Data Reduction Method. In: Tang – 18th International Conference on Pattern, 2006, pp. 556–559, DOI: 10.1109/ICPR.2006.1119.

[CLAR18] Clark, D.: Top 8 Python Machine Learning Libraries - KDnuggets. URL: https://www.kdnuggets.com/2018/10/top-python-machine-learning-libraries.html [Accessed: 05.08.2021]

[CODE21] Codecademy: Normalization | Codecademy. URL: https://www.codecademy.com/articles/normalization [Accessed: 28.07.2021]

[COHE14] Cohen, P.; West, S. G.; Aiken, L. S.: Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences: Psychology Press, 2014, ISBN: 9781135468255, DOI: 10.4324/9781410606266.

[CORA14] Coraddu, A.; Oneto, L.; Ghio, A.; Savio, S.; Anguita, D.; Figari, M.: Machine learning approaches for improving condition-based maintenance of naval propulsion plants. In: Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment, Vol. 230, 2014, No. 1, pp. 136–153, DOI: 10.1177/1475090214540874.

[CORT95] Cortes, C.; Vapnik, V.: Support-Vector Networks. In: Machine Learning, 1995, No. 20, pp. 273–297, DOI: 10.1007/BF00994018.

[CZIC08] Czichos, H.: Die Ingenieurwissenschaften - Ihr Profil in Technik und Gesellschaft: Springer Berlin Heidelberg, 2008, ISBN: 3540718524

[DABR20] Dabral, S.: What is an Outlier? How to handle and remove them? Algorithms that are affected by outliers. URL: https://medium.com/analytics-vidhya/what-is-an-outliers-how-to-detect-and-remove-them-which-algorithm-are-sensitive-towards-outliers-2d501993d59 [Accessed: 29.07.2021]

[DANG17] Dangeti, P.: Statistics for machine learning. Build supervised, unsupervised, and reinforcement learning models using both Python and R. Birmingham, UK: Packt Publishing, 2017, ISBN: 9781788291224

[DASA94] Dasarathy, B. V.: Minimal consistent set (MCS) identification for optimal nearest neighbor decision systems design. In: IEEE Transactions on Systems, Man, and Cybernetics, Vol. 24, 1994, No. 3, pp. 511–517, DOI: 10.1109/21.278999.

[DATA22a] Datacamp: Python for data science: Scikit-learn cheat sheet. URL: http://datacamp-community-prod.s3.amazonaws.com/eb807da5-dce5-4b97-a54d-74e89f14266b [Accessed: 28.04.2022]

[DATA22b] Datacamp: Python for data science: Data wrangling in pandas cheat sheet. URL: http://datacamp-community-prod.s3.amazonaws.com/d4efb29b-f9c6-4f1c-8c98-6f568d88b48f [Accessed: 28.04.2022]

[DAVI92] Davis, P. K.: Generalizing Concepts and Methods of Verification, Validation, and Accreditation (VV&A) for Military Simulations. Santa Monica, California, USA, 1992, ISBN: 0-8330-1298-3

[DEIS21] Deisenroth, M. P.; Faisal, A. A.; Ong, C. S.: Mathematics for Machine Learning: Cambridge University Press, 2021, ISBN: 9781108455145

[DEMP77] Dempster, A. P.; Laird, N. M.; Rubin, D. B.: Maximum Likelihood from Incomplete Data Via the EM Algorithm. In: Journal of the Royal Statistical Society: Series B (Methodological), Vol. 39, 1977, No. 1, pp. 1–22, DOI: 10.1111/j.2517-6161.1977.tb01600.x.

[DHAR13] Dhar, V.: Data Science and Prediction. In: Communications of the ACM, Vol. 56, 2013, No. 12, 64-73, DOI: 10.1145/2500499.

[DIAM09] Diamantini, C.; Potena, D.; Storti, E.: Ontology-Driven KDD Process Composition. In: Advances in Intelligent Data Analysis VIII, 8th International Symposium on Intelligent Data Analysis, 2009, DOI: 10.1007/978-3-642-03915-7_25.

[DIET00] Dietterich, T. G.: Ensemble Methods in Machine Learning. In: Multiple Classifier Systems. MCS 2000. Lecture Notes in Computer Science, Vol. 1857, 2000, pp. 1–15, DOI: 10.1007/3-540-45014-9_1.

[DOUZ18] Douzas, G.; Bacao, F.; Last, F.: Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE. In: Information Sciences, Vol. 465, 2018, pp. 1–20, DOI: 10.1016/j.ins.2018.06.056.

[DRUC96] Drucker, H.; Burges, C. J.; Kaufman, L.; Smola, A.; Vapnik, V.: Support Vector Regression Machines. In: Advances in Neural Information Processing Systems, Vol. 9, 1996

[DUA19] Dua, D.; Graff, C.: UCI Machine Learning Repository, 2019

[ELG21] Elg, E.: Jump Start Your Modeling with Random Forests. URL: https://www.elderresearch.com/blog/jump-start-your-modeling-with-random-forests/ [Accessed: 22.07.2021]

[ENGE96] Engels, R.: Planning tasks for Knowledge Discovery in Databases; Performing Task-Oriented User-Guidance. In: Proceedings of the International Conference on Knowledge Discovery and Data Mining, 1996, pp. 170–175

[ERIC20] Erickson, N.; Mueller, J.; Shirkov, A.; Zhang, H.; Larroy, P.; Li, M.; Smola, A.: AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data, 2020, DOI: 10.48550/arXiv.2003.06505.

[ESTE96] Ester, M.; Kriegel, H.-P.; Sander, J.; Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise, 1996

[FANC21] Fan, C.; Chen, M.; Wang, X.; Wang, J.; Huang, B.: A Review on Data Preprocessing Techniques Toward Efficient and Reliable Knowledge Discovery From Building Operational Data. URL: https://www.frontiersin.org/article/10.3389/fenrg.2021.652801 [Accessed: 24.08.2021]

[FARH07] Farhangfar, A.; Kurgan, L. A.; Pedrycz, W.: A Novel Framework for Imputation of Missing Values in Databases. In: IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, Vol. 37, 2007, No. 5, pp. 692–709, DOI: 10.1109/TSMCA.2007.902631.

[FAYY93] Fayyad, U.; Irani, K.: Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning, 1993

[FAYY96] Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P.: The KDD process for extracting useful knowledge from volumes of data. In: Communications of the ACM, Vol. 39, 1996, No. 11, pp. 27–34, DOI: 10.1145/240455.240464.

[FEIG82] Feigenbaum, E. A.: Knowledge Engineering for the 1980s. In: Stanford University Computer Science Department, 1982

[FERN05] Fernandez, J.-C.; Mounier, L.; Pachon, C.: A Model-Based Approach for Robustness Testing. In: Khendek, F., Dssouli, R. (eds) Testing of Communicating Systems. TestCom 2005. Lecture Notes in Computer Science, Vol. 3502, 2005, pp. 333–348, DOI: 10.1007/11430230_23.

[FERR94] Ferri, F. J.; Pudil, P.; Hatef, M.; Kittler, J.: Comparative study of techniques for large-scale feature selection. In: Gelsema, Kanal – Pattern Recognition in Practice IV, Vol. 16, 1994, pp. 403–413, DOI: 10.1016/B978-0-444-81892-8.50040-7.

[FEUR15] Feurer, M.; Klein, A.; Eggensperger, K.; Springenberg, J.; Blum, M.; Hutter, F.: Efficient and Robust Automated Machine Learning. In: Advances in Neural Information Processing Systems, Vol. 28, 2015

[FEUR19] Feurer, M.; Hutter, F.: Hyperparameter Optimization. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds) Automated Machine Learning. The Springer Series on Challenges in Machine Learning, 2019, pp. 3–33, DOI: 10.1007/978-3-030-05318-5_1.

[FEUR21] Feurer, M.; van Rijn, J. N.; Kadra, A.; Gijsbers, P.; Mallik, N.; Ravi, S.; Mueller, A.; Vanschoren, J.; Hutter, F.: OpenML-Python: an extensible Python API for OpenML. In: Journal of Machine Learning Research, Vol. 22, 2021, No. 100, DOI: 10.48550/arXiv.1911.02490.

[FEUR22] Feurer, M.; Eggensperger, K.; Falkner, S.; Lindauer, M.; Hutter, F.: Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning. In: Journal of Machine Learning Research, Vol. 23, 2022, No. 261, DOI: 10.48550/arXiv.2007.04074.

[FISH22] Fisher, R. A.: On the Mathematical Foundations of Theoretical Statistics. In: Kotz – Breakthroughs in statistics, 1922, pp. 11–44, DOI: 10.1007/978-1-4612-0919-5_2.

[FIX51] Fix, E.; Hodges, J. L.: Nonparametric discrimination: consistency properties, 1951

[FLAC12] Flach, P.: Machine learning. The art and science of algorithms that make sense of data. Cambridge: Cambridge Univ. Press, 2012, ISBN: 1107096391

[FREU97] Freund, Y.; Schapire, R. E.: A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. In: Journal of Computer and System Sciences, Vol. 55, 1997, No. 1, pp. 119–139, DOI: 10.1006/jcss.1997.1504.

[FRIE01] Friedman, J. H.: Greedy function approximation: A gradient boosting machine. In: The Annals of Statistics, Vol. 29, 2001, No. 5, DOI: 10.1214/aos/1013203451.

[FRIE10] Friedman, J.; Hastie, T.; Tibshirani, R.: Regularization Paths for Generalized Linear Models via Coordinate Descent. In: Journal of Statistical Software, Vol. 33, 2010, No. 1, pp. 1–22

[FROS19] Frost, J.: 5 Ways to Find Outliers in Your Data. URL: https://statisticsbyjim.com/basics/outliers/ [Accessed: 27.07.2021]

[FRYE19] Frye, M.; Schmitt, R. H.: Quality Improvement of Milling Processes Using Machine Learning-Algorithms. In: 16th IMEKO TC10 Conference “Testing, Diagnostics & Inspection as a comprehensive value chain for Quality & Safety", 2019

[FRYE20] Frye, M.; Schmitt, R. H.: Structured Data Preparation Pipeline for Machine Learning-Applications in Production. In: International Measurement Confederation (IMEKO) (ed.): 17th IMEKO TC 10 and EUROLAB Virtual Conference: “Global Trends in Testing, Diagnostics & Inspection for 2030”, 2020, pp. 241–246

[FRYE21a] Frye, M.; Mohren, J.; Schmitt, R. H.: Benchmarking of Data Preprocessing Methods for Machine Learning Applications in Production. In: 54th CIRP Conference on Manufacturing Systems, Procedia CIRP, Vol. 104, 2021, pp. 50–55, DOI: 10.1016/j.procir.2021.11.009.

[FRYE21b] Frye, M.; Krauß, J.; Schmitt, R. H.: Expert System for the Machine Learning Pipeline in Manufacturing. In: IFAC PapersOnLine, Vol. 54, 2021, No. 1, pp. 128–133, DOI: 10.1016/j.ifacol.2021.08.014.

[FRYE21c] Frye, M.; Gyulai, D.; Bergmann, J.; Schmitt, R. H.: Production rescheduling through product quality prediction. In: Procedia manufacturing, Vol. 54, 2021, pp. 142–147, DOI: 10.1016/j.promfg.2021.07.022.

[FUKU90] Fukunaga, K.: Introduction to statistical pattern recognition Series: Computer science and scientific computing. 2nd ed. Boston: Academic Press, 1990, ISBN: 9780080478654

[GABE17] Gabernet, A. R.: Breaking the 80/20 rule: How data catalogs transform data scientists’ productivity. URL: https://www.ibm.com/cloud/blog/ibm-data-catalog-data-scientists-productivity [Accessed: 24.07.2021]

[GAND21] Gandhi, A.: Data Augmentation | How to use Deep Learning when you have Limited Data. URL: https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/ [Accessed: 28.07.2021]

[GARC15] Garcia, S.; Luengo, J.; Herrera, F.: Data Preprocessing in Data Mining: Springer Cham Heidelberg New York Dordrecht London, 2015, ISBN: 978-3-319-10246-7, DOI: 10.1007/978-3-319-10247-4.

[GART18a] Gartner: How to Create a Business Case for Data Quality Improvement. URL: https://www.gartner.com/smarterwithgartner/how-to-create-a-business-case-for-data-quality-improvement/ [Accessed: 06.02.2022]

[GART18b] Gartner: Gartner Says Nearly Half of CIOs Are Planning to Deploy Artificial Intelligence. URL: https://www.gartner.com/en/newsroom/press-releases/2018-02-13-gartner-says-nearly-half-of-cios-are-planning-to-deploy-artificial-intelligence [Accessed: 06.02.2022]

[GATE72] Gates, G.: The reduced nearest neighbor rule (Corresp.). In: IEEE Transactions on Information Theory, Vol. 18, 1972, No. 3, pp. 431–433, DOI: 10.1109/TIT.1972.1054809.

[GE17] Ge, Z.; Song, Z.; Ding, S. X.; Huang, B.: Data Mining and Analytics in the Process Industry: The Role of Machine Learning. In: IEEE Access, Vol. 5, 2017, pp. 20590–20616, DOI: 10.1109/ACCESS.2017.2756872.

[GEIS17] Geissbauer, R.; Schrauf, S.; Berttram, P.: Digital Factories 2020: Shaping the future of manufacturing, 2017

[GELM06] Gelman, A.; Hill, J.: Data Analysis Using Regression and Multilevel/Hierarchical Models: Cambridge University Press, 2006, ISBN: 9781139460934

[GENE18] Genesis: Pros and Cons of K-Nearest Neighbors. URL: https://www.fromthegenesis.com/pros-and-cons-of-k-nearest-neighbors/ [Accessed: 22.07.2021]

[GÉRO18] Géron, A.: Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow. Konzepte, Tools und Techniken für intelligente Systeme. 1 ed.: O'Reilly, 2018, ISBN: 3960090617

[GEUR06] Geurts, P.; Ernst, D.; Wehenkel, L.: Extremely randomized trees. In: Machine Learning, Vol. 63, 2006, No. 1, pp. 3–42, DOI: 10.1007/s10994-006-6226-1.

[GHOJ19] Ghojogh, B.; Crowley, M.: The Theory Behind Overfitting, Cross Validation, Regularization, Bagging, and Boosting: Tutorial, 2019, DOI: 10.48550/arXiv.1905.12787.

[GILL21] Gillis, A. S.: Was ist Die sechs Vs von Big Data? URL: https://www.computerweekly.com/de/definition/Die-sechs-Vs-von-Big-Data [Accessed: 06.08.2021]

[GIOV21] Giovanelli, J.; Bilalli, B.; Abelló, A.: Effective data pre-processing for AutoML, 2021, pp. 1–10

[GIRA05] Giraud-Carrier, C.: The data mining advisor: meta-learning at the service of practitioners. In: International Conference on Machine Learning and Applications, 2005, DOI: 10.1109/ICMLA.2005.65.

[GODF16] Godfrey, P.; Gryz, J.; Lasek, P.: Interactive Visualization of Large Data Sets. In: IEEE Transactions on Knowledge and Data Engineering, Vol. 28, 2016, No. 8, pp. 2142–2157, DOI: 10.1109/TKDE.2016.2557324.

[GONZ20] González Rodríguez, G.; Gonzalez-Cava, J. M.; Méndez Pérez, J. A.: An intelligent decision support system for production planning based

on machine learning. In: Journal of Intelligent Manufacturing, Vol. 31, 2020, No. 5, pp. 1257–1273, DOI: 10.1007/s10845-019-01510-y.

[GOOD16] Goodfellow, I.; Bengio, Y.; Courville, A.: Deep learning. Cambridge, Massachusetts, London, England: MIT Press, 2016, ISBN: 9780262035613

[GOOG20] Google LLC: AutoML Tables Dokumentation. Datenvorbereitung, die AutoML Tables erledigt. URL: https://cloud.google.com/automl-tables/docs/data-best-practices?hl=de [Accessed: 28.04.2022]

[GOSW17] Goswami, S.; Chakraborty, S.; Saha, H. N.: An Univariate Feature Elimination Strategy for Clustering Based on Metafeatures. In: International Journal of Intelligent Systems and Applications, Vol. 9, 2017, No. 10, pp. 20–30, DOI: 10.5815/ijisa.2017.10.03.

[GRUB50] Grubbs, F. E.: Sample Criteria for Testing Outlying Observations. In: The Annals of Mathematical Statistics, Vol. 21, 1950, No. 1, pp. 27–58, DOI: 10.1214/aoms/1177729885.

[GUPA19] Gupta, R.: An Introduction to Discretization Techniques for Data Scientists. URL: https://towardsdatascience.com/an-introduction-to-discretization-in-data-science-55ef8c9775a2 [Accessed: 07.08.2021]

[GUPT17] Gupta, A.: How to read most commonly used file formats in Data Science (using Python)? URL: https://www.analyticsvidhya.com/blog/2017/03/read-commonly-used-formats-using-python/ [Accessed: 06.08.2021]

[GURS16] Gursch H.; Wuttei A.; Gangloff T.: Learning Systems for Manufacturing Management Support. In: SamI40 workshop at i-KNOW, 2016

[GUYO02] Guyon, I.; Weston, J.; Barnhill, S.; Vapnik, V.: Gene Selection for Cancer Classification using Support Vector Machines. In: Machine Learning, Vol. 46, 2002, No. 1/3, pp. 389–422, DOI: 10.1023/A:1012487302797.

[GUYO03] Guyon, I.; Elisseeff, A.: An Introduction to Variable and Feature Selection. In: Journal of Machine Learning Research, Vol. 3, 2003, pp. 1157–1182

[GYUL18] Gyulai, D.; Pfeiffer, A.; Bergmann, J.; Gallina, V.: Online lead time prediction supporting situation-aware production control. In: Procedia CIRP, 2018, No. 78, pp. 190–195, DOI: 10.1016/j.procir.2018.09.071.

[H2O16] H2O.ai: Python Interface for H2O, Python module version 3.10.0.8. URL: https://github.com/h2oai/h2o-3 [Accessed: 28.04.2022]

[HAET99] Haettenschwiler, P.: Neues anwenderfreundliches Konzept der Entscheidungsunterstützung. In: Gutes Entscheiden in Wirtschaft, Politik and Gesllschaft, 1999, pp. 189–208

[HAIB08] Haibo He; Yang Bai; E. A. Garcia; Shutao Li: ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In: IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), 2008, pp. 1322–1328, DOI: 10.1109/IJCNN.2008.4633969.

[HAID20] Haider, S. N.; Zhao, Q.; Meran, B. K.: Automated data cleaning for data centers: A case study. In: Proceedings of the 39th Chinese Control Conference, 2020, pp. 3227–3232, DOI: 10.23919/CCC50068.2020.9189357.

[HAIF19] Haifeng Jin; Qingquan Song; Xia Hu: Auto-Keras: An Efficient Neural Architecture Search System, 2019, DOI: 10.48550/arXiv.1806.10282.

[HALE18] Hale, J.: Smarter Ways to Encode Categorical Data for Machine Learning. URL: https://towardsdatascience.com/smarter-ways-to-encode-categorical-data-for-machine-learning-part-1-of-3-6dca2f71b159 [Accessed: 27.07.2021]

[HALK11] Halko, N.; Martinsson, P.-G.; Tropp, J. A.: Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. In: SIAM Review, Vol. 53, 2011, No. 2, DOI: 10.1137/090771806.

[HALL09] Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; Witten, I. H.: The WEKA data mining software. In: ACM SIGKDD Explorations Newsletter, Vol. 11, 2009, No. 1, pp. 10–18, DOI: 10.1145/1656274.1656278.

[HAN05] Han, H.; Wang, W.-Y.; Mao, B.-H.: Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In: Huang, DS., Zhang, XP., Huang, GB. (eds) Advances in Intelligent Computing. ICIC 2005. Lecture Notes in Computer Science, Vol. 3644, 2005, pp. 878–887, DOI: 10.1007/11538059_91.

[HAN12] Han, J.; Pei, J.; Kamber, M.: Data mining. Concepts and techniques Series: The Morgan Kaufmann series in data management systems. 3rd ed. Amsterdam, Boston: Elsevier/Morgan Kaufmann, 2012, ISBN: 9780123814807

[HANS87] Hanson, S.; Burr, D.: Minkowski-r back-propagation: Learning in connectionist models with non-euclidian error signals. In: Neural information processing systems, 1987

[HARD06] Harding, J. A.; Shahbaz M.; Srinivas: Data Mining in Manufacturing: A Review. In: Journal of Manufacturing Science Engineering, Vol. 128, 2006, No. 4, pp. 969–976, DOI: 10.1115/1.2194554.

[HART68] Hart, P.: The condensed nearest neighbor rule (Corresp.). In: IEEE Transactions on Information Theory, Vol. 14, 1968, No. 3, pp. 515–516, DOI: 10.1109/TIT.1968.1054155.

[HARV21] Harvard Business Manager: Getting AI to Scale, 2021

[HECH04] Hechenbichler, K.; Schliep, K.: Weighted k-Nearest-Neighbor Techniques and Ordinal Classification, 2004, DOI: 10.5282/ubm/epub.1769.

[HELW15] Helwig, N.; Pignanelli, E.; Schütze, A.: Condition monitoring of a complex hydraulic system using multivariate statistics. In: IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings, 2015, pp. 210–215, DOI: 10.1109/I2MTC.2015.7151267.

[HILB67] Hilborn, C. G.; Lainiotis, D.: The Condensed Nearest Neighbor Rule, 1967

[HINT02] Hinton, G. E.; Roweis, S.: Stochastic Neighbor Embedding, 2002

[HOER70] Hoerl, A. E.; Kennard, R. W.: Ridge Regression: Biased Estimation for Nonorthogonal Problems. In: Technometrics, Vol. 12, 1970, No. 1, pp. 55–67, DOI: 10.1080/00401706.1970.10488634.

[HOLS21] Holst, C.; Königs, M.; Garcia, E. M.; Ganser, P.; Bergs, T.: Spatially Resolved Tool Wear Prediction in Finish Milling. In: Procedia CIRP, 2021, No. 104, pp. 85–90, DOI: 10.1016/j.procir.2021.11.015.

[HOLT93] Holte, R. C.: Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. In: Machine Learning, Vol. 11, 1993, No. 1, pp. 63–90, DOI: 10.1023/A:1022631118932.

[HRAN16] Hranisavljevic, N.; Niggemann, O.; Maier, A.: A Novel Anomaly Detection Algorithm for Hybrid Production Systems based on Deep Learning and Timed Automata. In: The 27th International Workshop on Principles of Diagnosis: DX-2016, 2016

[HUBE64] Huber, P. J.: Robust estimation of a location parameter Series: Annals Mathematics Statistics, 35, 1964, DOI: 10.1214/aoms/1177703732.

[HUGH68] Hughes, G.: On the mean accuracy of statistical pattern recognizers. In: IEEE Transactions on Information Theory, Vol. 14, 1968, No. 1, pp. 55–63, DOI: 10.1109/TIT.1968.1054102.

[HULS07] van Hulse, J. D.; Khoshgoftaar, T. M.; Huang, H.: The pairwise attribute noise detection algorithm. In: Knowledge and Information Systems, Vol. 11, 2007, No. 2, pp. 171–190, DOI: 10.1007/s10115-006-0022-x.

[HUTT19] Hutter, F.; Kotthoff, L.; Vanschoren, J. (ed.): Automated Machine Learning Series: The Springer series on challenges in machine learning. Cham: Springer International Publishing, 2019, DOI: 10.1007/978-3-030-05318-5.

[HYVÄ00] Hyvärinen, A.; Oja, E.: Independent component analysis: algorithms and applications. In: Neural Networks, Vol. 13, 2000, No. 4-5, pp. 411–430, DOI: 10.1016/S0893-6080(00)00026-5.

[ICTM21] ICTM: Optimized Blisk Series Production by Use of Tool Path Compensation Methods, 2021

[INIT18] inIT - Institute Industrial IT and Ostwestfalen-Lippe University of Applied Sciences: High Storage System Data for Energy Optimization. URL: https://www.kaggle.com/inIT-OWL/high-storage-system-data-for-energy-optimization [Accessed: 08.08.2021]

[IPT21] IPT: Mit Künstlicher Intelligenz in den Weltraum: Sensoren und selbstlernende Systeme verbessern Herstellungsverfahren für die Ariane 6, 2021

[ISO08] ISO / IEC 25012:2008: Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Data quality model, 2008

[JAME61] James, W.; Stein, C.: Estimation with Quadratic Loss. In: Proceedings of the Fourth Berkeley, 1961, pp. 361–380

[JESS91] Jessup, L. M.; Tansik, D. A.: Decision Making in an Automated Environment: The Effects of Anonymity and Proximity with a Group Decision Support System. In: Decision Sciences, Vol. 22, 1991, No. 2, pp. 266–279, DOI: 10.1111/j.1540-5915.1991.tb00346.x.

[JOHN95] John, G. H.: Robust Decision Trees: Removing Outliers from Databases. In: KDD '95: Proceedings of the First International Conference on Knowledge Discovery and Data Mining, 1995, pp. 174–179, DOI: 10.5555/3001335.3001364.

[JOLL02] Jolliffe, I. T.: Principal Component Analysis Series: Springer series in statistics. Second Edition ed. New York, NY: Springer-Verlag New York Inc, 2002, ISBN: 9780387224404, DOI: 10.1007/b98835.

[JOLL16] Jolliffe, I. T.; Cadima, J.: Principal component analysis: a review and recent developments. In: Philosophical transactions. Series A, Mathematical, physical, and engineering sciences, Vol. 374, 2016, No. 2065, DOI: 10.1098/rsta.2015.0202.

[JUTT91] Jutten, C.; Herault, J.: Blind separation of sources, part I: An adaptive algorithm based on neuromimetic architecture. In: Signal Processing, Vol. 24, 1991, No. 1, pp. 1–10, DOI: 10.1016/0165-1684(91)90079-X.

[KAGG19] Kaggle: Vega shrink-wrapper component degradation. New vs worn cutting blade data. URL: https://www.kaggle.com/datasets/inIT-OWL/vega-shrinkwrapper-runtofailure-data/metadata [Accessed: 08.08.2021]

[KAPR20] Kapri, A.: PCA vs LDA vs T-SNE — Let's Understand the difference between them! URL: https://medium.com/analytics-vidhya/pca-vs-lda-vs-t-sne-lets-understand-the-difference-between-them-22fa6b9be9d0 [Accessed: 08.08.2021]

[KATA20] Katara, V.: Dimensionality Reduction — PCA vs LDA vs t-SNE - Analytics Vidhya - Medium. URL: https://medium.com/analytics-vidhya/dimensionality-reduction-pca-vs-lda-vs-t-sne-681636bc686 [Accessed: 08.08.2021]

[KE17] Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; Liu, T.: LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In: Advances in Neural Information Processing Systems, Vol. 30, 2017, pp. 3149–3157, DOI: 10.5555/3294996.3295074.

[KEEN80] Keen, P.: Decision Support systems: A research perspective, 1980

[KELL15] Kelleher, J. D.; MacNamee, B.; D'Arcy, A.: Fundamentals of machine learning for predictive data analytics. Algorithms, worked examples, and case studies. Cambridge, Massachusetts, London, England: The MIT Press, 2015, ISBN: 9780262029445

[KERB92] Kerber R.: Chimerge: Discretization of numeric attributes. In: Proceedings of the tenth national conference on Artificial intelligence, 1992, pp. 123–128, DOI: 10.5555/1867135.1867154.

[KHAN20] Khan, S. I.; Hoque, Abu Sayed Md Latiful: SICE: an improved missing data imputation technique. In: Journal of Big Data, Vol. 7, 2020, No. 1, p.37, DOI: 10.1186/s40537-020-00313-w.

[KHOS06] Khoshgoftaar, T. M.; JOSHI, V.; SELIYA, N.: Detecting Noisy Instances With The Ensemble Filter: A Study In Software Quality Estimation. In: International Journal of Software Engineering and Knowledge Engineering, Vol. 16, 2006, No. 1, pp. 53–76, DOI: 10.1142/S0218194006002677.

[KHOS20] Khosravi, P.; Vergari, A.; Choi, Y.; Liang, Y.; van den Broeck, G.: Handling Missing Data in Decision Trees: A Probabilistic Approach. In: Computer Science, Machine Learning, 2020, DOI: 10.48550/arXiv.2006.16341.

[KIBE20] Kibet, A.: Classification in Imbalanced Data Sets. - Towards Data Science. URL: https://towardsdatascience.com/classification-framework-for-imbalanced-data-9a7961354033 [Accessed: 29.07.2021]

[KIBL87] KIBLER, D.; AHA, D. W.: Learning Representative Exemplars of Concepts: An Initial Case Study. In: Proceedings of the Fourth International Workshop on Machine Learning, 1987, pp. 24–30, DOI: 10.1016/B978-0-934613-41-5.50006-4.

[KIES19] Kiesel, R.; Dering, J.; Schmitt, R. H.: Cybersecurity in der vernetzten Produktion. In: Fabriksoftware, Vol. 24, 2019, No. 4, pp. 21–24, DOI: 10.30844/FS19-4_21-24.

[KIM05] Kim, H.; Golub, G. H.; Park, H.: Missing value estimation for DNA microarray gene expression data: local least squares imputation. In: Bioinformatics (Oxford, England), Vol. 21, 2005, No. 2, pp. 187–198, DOI: 10.1093/bioinformatics/bth499.

[KIM07] Kim, S.-J.; Koh, K.; Lustig, M.; Boyd, S.: An Interior-Point Method for Large-Scale l1-Regularized Least Squares. In: IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING, Vol. 1, 2007, No. 4, pp. 606–617, DOI: 10.1109/JSTSP.2007.910971.

[KIM16] Kim, G.; Humble, J.; Debois, P.; Willis, J.: The DevOps Handbook: O'Reilly, 2016, ISBN: 9781942788003

[KIM18] Kim, D.-H.; Kim, T. J. Y.; Wang, X.; Kim, M.; Quan, Y.-J.; Oh, J. W.; Min, S.-H.; Kim, H.; Bhandari, B.; Yang, I.; Ahn, S.-H.: Smart Machining Process Using Machine Learning: A Review and Perspective on Machining Industry. In: International Journal of Precision Engineering and Manufacturing-Green Technology, Vol. 5, 2018, No. 4, pp. 555–568, DOI: 10.1007/s40684-018-0057-y.

[KLIM21] Klimovic, A.: Rethinking Data Storage and Preprocessing for ML. URL: https://www.sigarch.org/rethinking-data-storage-and-preprocessing-for-ml/ [Accessed: 10.08.2021]

[KLOC08] Klocke, F.; König, W.: Fertigungsverfahren 1. Drehen, Fräsen, Bohren Series: VDI-Buch. 8th ed. Berlin, Heidelberg: Springer-Verlag, 2008, ISBN: 9783540358343, DOI: 10.1007/978-3-540-35834-3.

[KLOC15] Klocke, F.; Schmitt, R.; Zeis, M.; Heidemanns, L.; Kerkhoff, J.; Heinen, D.; Klink, A.: Technological and Economical Assessment of Alternative Process Chains for Blisk Manufacture. In: Procedia CIRP, Vol. 35, 2015, pp. 67–72, DOI: 10.1016/j.procir.2015.08.052.

[KOHA96] Kohavi, R.; Wolpert, D. H.: Bias Plus Variance Decomposition for Zero-One Loss Functions. In: Proceedings of the 13th International Conference for Machine Learning, 1996, DOI: 10.5555/3091696.3091730.

[KOHO82] Kohonen, T.: Self-organized formation of topologically correct feature maps. In: Biological Cybernetics, Vol. 43, 1982, No. 1, pp. 59–69, DOI: 10.1007/BF00337288.

[KÖKS11] Köksal, G.; Batmaz, I.; Testik, M. C.: A review of data mining applications for quality improvement in manufacturing industry. In: Expert Systems with Applications, Vol. 38, 2011, No. 10, pp. 13448–13467, DOI: 10.1016/j.eswa.2011.04.063.

[KOLC05] Kolcz, A.; Chowdhury, A.: Improved Naive Bayes for Extremely Skewed Misclassification Costs. In: Jorge, A.M., Torgo, L., Brazdil, P., Camacho, R., Gama, J. (eds) Knowledge Discovery in Databases: PKDD 2005. PKDD 2005. Lecture Notes in Computer Science, Vol. 3721, 2005, pp. 561–568, DOI: 10.1007/11564126_58.

[KOME19] Komer, B.; Bergstra, J.; Eliasmith, C.: Hyperopt-Sklearn. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds) Automated Machine Learning. The Springer Series on Challenges in Machine Learning, 2019, pp. 97–111, DOI: 10.1007/978-3-030-05318-5_5.

[KOTS06] Kotsiantis, S. B.; Kanellopoulos, D.; Pintelas, P. E.: Data Preprocessing for Supervised Learning. In: International Journal of Computer Science, 2006, pp. 111–117

[KOTS07] Kotsiantis, S.: Supervised Machine Learning: A Review of Classification Techniques. In: Proceedings of the 2007 conference on Emerging Artificial Intelligence Applications in Computer Engineering, Vol. 31, 2007, pp. 249–268, DOI: 10.5555/1566770.1566773.

[KOTT13] Kotthoff, L.; Thornton, C.; Hutter, F.: User Guide for Auto-WEKA version 2.6. URL: https://www.cs.ubc.ca/labs/algorithms/Projects/autoweka/manual.pdf [Accessed: 14.07.2021]

[KRAU19a] Krauß, J.; Frye, M.; Döhler Beck, G. T.; Schmitt, R. H.: Selection and Application of Machine Learning-Algorithms in Production Quality. In: Beyerer, J., Kühnert, C., Niggemann, O. (eds) Machine Learning for Cyber Physical Systems. Technologien für die intelligente Automation, Vol. 9, 2019, pp. 46–57, DOI: 10.1007/978-3-662-58485-9_6.

[KRAU19b] Krauß, J.; Dorißen, J.; Mende, H.; Frye, M.; Schmitt, R. H.: Machine Learning and Artificial Intelligence in Production: Application Areas and Publicly Available Data Sets. In: Wulfsberg, J.P., Hintze, W., Behrens, BA. (eds) Production at the leading edge of technology, 2019, DOI: 10.1007/978-3-662-60417-5_49.

[KRAU20] Krauß, J.: AutoML Benchmark in Production. URL: https://jonathankrauss.github.io/AutoML-Benchmark/ [Accessed: 09.08.2021]

[KRAU22] Krauß, J.: Optimizing Hyperparameters for Machine Learning Algorithms in Production: Apprimus Verlag, 2022, ISBN: 978-3-98555-020-3

[KRIE08] Kriegel, H.-P.; Hubert, M.; Zimek, A.: Angle-based outlier detection in high-dimensional data, 2008, p.444, DOI: 10.1145/1401890.1401946.

[KRIS15] Krishnan, S.; Wang. Jiannan; Franklin, M. J.; Goldberg, K.; Kraska, T.; Milo, T.; Wu, E.: SampleClean: Fast and Reliable Analytics on Dirty Data. In: Computer Science, IEEE Data Engineering Bulletin, 2015

[KRIS16] Krishnan, S.; Wang, J.; Wu, E.; Franklin, M. J.; Goldberg, K.: ActiveClean. In: Proceedings of the VLDB Endowment, Vol. 9, 2016, No. 12, pp. 948–959, DOI: 10.14778/2994509.2994514.

[KRIS17] Krishnan, S.; Franklin, M. J.; Goldberg, K.; Wu, E.: BoostClean: Automated Error Detection and Repair for Machine Learning. In: Computer Science, 2017

[KRUS64] Kruskal, J. B.: Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. In: Psychometrika, Vol. 29, 1964, No. 1, pp. 1–27, DOI: 10.1007/BF02289565.

[KUBA97] Kubat, M.; Matwin, S.: Addressing the Curse of Imbalanced Training Sets: One-Sided Selection. In: Proceedings of the Fourteenth International Conference on Machine Learning, 1997, pp. 179–186

[KUBI77] Kubicek, H.: Heuristische Bezugsrahmen und heuristisch angelegte Forschungsdesigns als Elemente einer Konstruktionsstrategie empirischer Forschung. In: Köhler, R. (ed.): Empirische und handlungstheoretische Forschungskonzeptionen in der Betriebswirtschaftslehre. Bericht über d. Tagung in Aachen, 1977, pp. 3–36

[KUHN18] Kuhn, M.; Johnson, K.: Applied predictive modeling: Springer New York, 2018, DOI: 10.1007/978-1-4614-6849-3.

[KUHN20] Kuhn, M.; Johnson, K.: Feature engineering and selection. A practical approach for predictive models Series: Chapman & Hall/CRC data

science series. Boca Raton: CRC Press Taylor & Francis Group, 2020, ISBN: 9781138079229

[KUMA20] Kumar, A.; Jain, M.: Ensemble Learning for AI Developers. Learn Bagging, Stacking, and Boosting Methods with Use Cases: Apress, 2020, DOI: 10.1007/978-1-4842-5940-5.

[KUNZ19] Kunz, N.: smogn/smogn_example_2_int.ipynb at master · nickkunz/smogn. URL: https://github.com/nickkunz/smogn/blob/master/examples/smogn_example_2_int.ipynb [Accessed: 16.08.2021]

[KUNZ20] Kunz, N.: GitHub - nickkunz/smogn: Synthetic Minority Over-Sampling Technique for Regression. URL: https://github.com/nickkunz/smogn/blob/master/smogn/smoter.py [Accessed: 09.08.2021]

[KURS10] Kursa, M. B.; Rudnicki, W. R.: Feature Selection with the Boruta Package. In: Journal of Statistical Software, Vol. 36, 2010, No. 11, DOI: 10.18637/jss.v036.i11.

[LAAD19] Laadan, D.; Vainshtein, R.; Curiel, Y.; Katz, G.; Rokach, L.: RankML: a Meta Learning-Based Approach for Pre-Ranking Machine Learning Pipelines. In: Computer Science, Machine Learning, 2019, pp. 1–8, DOI: 10.48550/arXiv.1911.00108.

[LANG18] Langlitz, L.; Gallina, V.; Ansari, F.; Gyulai, D.; Pfeiffer, A.; Sihn, W.; Monostori; László: Lead time prediction using machine learning algorithms: A case study by a semiconductor manufacturer. In: Procedia CIRP, Vol. 72, 2018, pp. 1051–1056, DOI: 10.1016/j.procir.2018.03.148.

[LARI20] Larionov, M.: Target Encoding and Bayesian Target Encoding - Towards Data Science. URL: https://towardsdatascience.com/target-encoding-and-bayesian-target-encoding-5c6a6c58ae8c [Accessed: 28.07.2021]

[LAUR01] Laurikkala, J.: Improving Identification of Difficult Small Classes by Balancing Class Distribution. In: Quaglini, S., Barahona, P., Andreassen, S. (eds) Artificial Intelligence in Medicine. AIME 2001. Lecture Notes in Computer Science, Vol. 2101, 2001, pp. 63–66, DOI: 10.1007/3-540-48229-6_9.

[LAUS08] Lausser, L.; Buchholz, M.; Kestler, H. A.: Boosting Threshold Classifiers for High- Dimensional Data in Functional Genomics. In: Prevost, L., Marinai, S., Schwenker, F. (eds) Artificial Neural Networks in Pattern Recognition. ANNPR 2008. Lecture Notes in Computer Science, Vol. 5064, 2008, pp. 147–156, DOI: 10.1007/978-3-540-69939-2_15.

[LAZA05] Lazarevic, A.; Kumar, V.: Feature bagging for outlier detection. In: Proceedings of the eleventh ACM, 2005, p.157, DOI: 10.1145/1081870.1081891.

[LEDE20] LeDell, E.; Poirier, S.: H2O AutoML: Scalable Automatic Machine Learning. In: 7th ICML Workshop on Automated Machine Learning, 2020, pp. 1–16

[LETR20] Le, T. T.; Fu, W.; Moore, J. H.: Scaling tree-based automated machine learning to biomedical big data with a feature set selector. In: Bioinformatics (Oxford, England), Vol. 36, 2020, No. 1, pp. 250–256, DOI: 10.1093/bioinformatics/btz470.

[LEWI46] Lewin, K.: Action Research and Minority Problems. In: Journal of Social Issues, Vol. 2, 1946, No. 4, pp. 34–46, DOI: 10.1111/j.1540-4560.1946.tb02295.x.

[LIND19] Lindemann, B.; Fesenmayr, F.; Jazdi, N.; Weyrich, M.: Anomaly detection in discrete manufacturing using self-learning approaches. In: Procedia CIRP, Vol. 79, 2019, pp. 313–318, DOI: 10.1016/j.procir.2019.02.073.

[LIND99] Lindner, G.; Studer, R.: AST: Support for Algorithm Selection with a CBR approach. In: Żytkow, J.M., Rauch, J. (eds) Principles of Data Mining and Knowledge Discovery. PKDD 1999. Lecture Notes in Computer Science, Vol. 1704, 1999, pp. 418–423, DOI: 10.1007/978-3-540-48247-5_52.

[LITT88] Little, R. J. A.: A Test of Missing Completely at Random for Multivariate Data with Missing Values. In: Journal of the American Statistical Association, Vol. 83, 1988, No. 404, p.1198, DOI: 10.2307/2290157.

[LIU08a] Liu, F. T.; Ting, K. M.; Zhou, Z.-H.: Isolation Forest. In: Giannotti: IEEE International Conference, 2008, pp. 413–422, DOI: 10.1109/ICDM.2008.17.

[LIU08b] Liu, H.; Motoda, H.: Computational Methods of Feature Selection. Boca Raton, London, New York: Taylor & Francis, 2008, ISBN: 978-1-58488-878-9

[LIU95] Liu, H.; Setiono, R.: Chi2: feature selection and discretization of numeric attributes. In: IEEE Computer Society - Proceedings, 1995, pp. 388–391, DOI: 10.1109/TAI.1995.479783.

[LIU96a] Liu, H.; Setiono, R.: A Probabilistic Approach to Feature Selection - A Filter Solution. In: ICML'96: Proceedings of the Thirteenth International Conference on International Conference on Machine Learning, 1996, pp. 319–327, DOI: 10.5555/3091696.3091735.

[LIU96b] Liu, H.; Setiono, R.: Feature Selection And Classification - A Probabilistic Wrapper Approach. In: IEA/AIE'96: Proceedings of the 9th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, 1996, pp. 419–424, DOI: 10.5555/3104635.3104707.

[LIUY20] Liu, Y.; Mu, Y.; Chen, K.; Li, Y.; Guo, J.: Daily Activity Feature Selection in Smart Homes Based on Pearson Correlation Coefficient. In: Neural Processing Letters, Vol. 51, 2020, No. 2, pp. 1771–1787, DOI: 10.1007/s11063-019-10185-8.

[LUEN11] Luengo, J.; Fernández, A.; García, S.; Herrera, F.: Addressing data complexity for imbalanced data sets: analysis of SMOTE-based oversampling and evolutionary undersampling. In: Soft Computing, Vol. 15, 2011, No. 10, pp. 1909–1936

[LUEN18] Luengo, J.; Shim, S.-O.; Alshomrani, S.; Altalhi, A.; Herrera, F.: CNC-NOS: Class noise cleaning by ensemble filtering and noise scoring. In: Knowledge-Based Systems, Vol. 140, 2018, pp. 27–49, DOI: 10.1016/j.knosys.2017.10.026.

[LUEN21] Luengo, J.; Sánchez-Tarragó, D.; Prati, R. C.; Herrera, F.: Multiple instance classification: Bag noise filtering for negative instance noise cleaning. In: Information Sciences, Vol. 579, 2021, pp. 388–400, DOI: 10.1016/j.ins.2021.07.076.

[LYAS20] Lyaschenko, V.: Data Augmentation in Python: Everything You Need to Know. URL: https://neptune.ai/blog/data-augmentation-in-python [Accessed: 28.07.2021]

[LYON16] Lyon, R. J.; Stappers, B. W.; Cooper, S.; Brooke, J. M.; Knowles, J. D.: Fifty years of pulsar candidate selection: from simple filters to a new principled real-time classification approach. In: Monthly Notices of the Royal Astronomical Society, Vol. 459, 2016, No. 1, pp. 1104–1123, DOI: 10.1093/mnras/stw656.

[MAAT08] van der Maaten, L.; Hinton, G.: Visualizing Data using t-SNE. In: Journal of Machine Learning Research, Vol. 9, 2008, No. 86, pp. 2579–2605

[MACQ67] MacQueen, J.; Le Cam, L. M.; Neyman, J.: Some Methods for Classification and Analysis of Multivariate Observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability: Weather modification, 1967, pp. 281–297

[MAGA21] Magaga, A. M.: Identifying, Cleaning and replacing outliers | Titanic Dataset. URL: https://medium.com/analytics-vidhya/identifying-cleaning-and-replacing-outliers-titanic-dataset-20182a062893 [Accessed: 30.08.2021]

[MAHD19] Mahdavi, M.; Neutatz, F.; Visengeriyeva, L.; Abedjan, Z.: Towards Automated Data Cleaning Workflows, 2019

[MANI03] Mani, I.; Zhang, J. P.: kNN approach to unbalanced data distributions: a case study involving information extraction. In: Proceeding of International Conference on Machine Learning (ICML 2003), Workshop on Learning from Imbalanced Data Sets, 2003

[MARA99] Marakas, G. M.: Decision Support Systems in the 21st Century, 1999, ISBN: 9780137441860

[MAYR02] Mayring, P.: Einführung in die qualitative Sozialforschung: Beltz Verlagsgruppe, 2002, ISBN: 9783407290939

[MCIN18] McInnes, L.; Healy, J.: UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction, 2018, DOI: 10.48550/arXiv.1802.03426.

[MCKI16] McKinsey Global Institute: The Age of Analytics: Competing in a Data-Driven World. URL: https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-age-of-analytics-competing-in-a-data-driven-world [Accessed: 08.08.2021]

[MCKI17] McKinsey & Company: Smartening up with Artificial Intelligence (AI): What's in it for Germany and its Industrial Sector?, 2017

[MCKI21] McKinsey & Company: Scaling AI in the sector that enables it lessons for semiconductor device makers. URL: https://www.mckinsey.de/industries/semiconductors/our-insights/scaling-ai-in-the-sector-that-enables-it-lessons-for-semiconductor-device-makers [Accessed: 08.08.2021]

[MENA14] Menardi, G.; Torelli, N.: Training and assessing classification rules with imbalanced data. In: Data Mining and Knowledge Discovery, Vol. 28, 2014, No. 1, pp. 92–122, DOI: 10.1007/s10618-012-0295-5.

[MERC20] Mercedes Benz: Mercedes-Benz Greener Manufacturing. URL: https://www.kaggle.com/datasets/philanipro/mercedesbenz-greener-manufacturing/metadata [Accessed: 10-Feb-22]

[MICC01] Micci-Barreca, D.: A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. In: ACM SIGKDD Explorations Newsletter, Vol. 3, 2001, No. 1, pp. 27–32, DOI: 10.1145/507533.507538.

[MICH94] Michie, D.; Spiegelhalter, D. J.; Taylor, C. C.: Machine Learning, Neural and Statistical Classification, 1994, DOI: 10.2307/1269742.

[MICR20] Microsoft Corporation: Azure Machine Learning Documentation. URL: https://docs.microsoft.com/de-de/azure/machine-learning/ [Accessed: 08.09.2021]

[MICR22] Microsoft Corporation: LightGBM. URL: https://lightgbm.readthedocs.io/en/v3.3.2/ [Accessed: 08.08.2021]

[MIER06] Mierswa, I.; Wurst, M.; Klinkenberg, Ralf, Scholz, Martin; Euler, T.: YALE: rapid prototyping for complex data mining tasks. In: 12th ACM SIGKDD international conference on Knowledge discovery and data mining, 2006, pp. 935–940, DOI: 10.1145/1150402.1150531.

[MINO15] Minoufekr, M.: Macroscopic simulation of multi-axis machining processes Series: Edition Wissenschaft. 1st ed. Aachen: Apprimus-Verlag, 2015, ISBN: 9783863593032

[MITC97] Mitchell, T.: Machine Learning. New York, 1997, ISBN: 9780070428072

[MOLD17] Moldovan, D.; Cioara, T.; Anghel, I.; Salomie, I.: Machine learning for sensor-based manufacturing processes. In: IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), 2017, pp. 147–154, DOI: 10.1109/ICCP.2017.8116997.

[MOLI19] Molino, P.; Dudin, Y.; Miryala, S. S.: Ludwig: a type-based declarative deep learning toolbox, 2019, DOI: 10.48550/arXiv.1909.07930.

[MOLL02] Mollineda, R. A.; Ferri, F. J.; Vidal, E.: An efficient prototype merging strategy for the condensed 1-NN rule through class-conditional hierarchical clustering. In: Pattern Recognition, Vol. 35, 2002, No. 12, pp. 2771–2782, DOI: 10.1016/S0031-3203(01)00208-4.

[MOLN22] Molnar, C.: Interpretable Machine Learning: A Guide for Making Black Box Models Explainable. URL: christophm.github.io/interpretable-ml-book/

[MOOE21] Moore, K.; Kan, K. F.; McGuire, L.; Tovbin, M.; Ovsiankin, M.; Loh, M.; Weil, M.; Nabar, S.; Gordon, V.; Patryshev, V.: TransmogrifAI. URL: https://github.com/salesforce/TransmogrifAI [Accessed: 14.07.2021]

[MORI02] Morik, K.; Scholz, M.: The MiningMart Approach. In: Informatik bewegt: Informatik 2002 - 32. Jahrestagung der Gesellschaft für Informatik e.v. (GI), 2002, pp. 811–818

[MUHL04] Muhlenbach, F.; Lallich, S.; Zighed, D. A.: Identifying and Handling Mislabelled Instances. In: Journal of Intelligent Information Systems, Vol. 22, 2004, No. 1, pp. 89–109, DOI: 10.1023/A:1025832930864.

[MURP12] Murphy, K. P.: Machine learning. A probabilistic perspective Series: Adaptive computation and machine learning series. Cambridge, Mass.: MIT Press, 2012, ISBN: 9780262305242

[NASA21] Nasa: Prognostics Center of Excellence - Data Repository. URL: https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/ [Accessed: 07.08.2021]

[NGUY11] Nguyen, H. M.; Cooper, E. W.; Kamei, K.: Borderline over-sampling for imbalanced data classification. In: International Journal of Knowledge Engineering and Soft Data Paradigms, Vol. 3, 2011, No. 1, p.4, DOI: 10.1504/IJKESDP.2011.039875.

[NUMP21] Numpy: numpy.random.normal — NumPy v1.21 Manual. URL: https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html [Accessed: 13.08.2021]

[OKUD19] Okudan, A.: 3D Printer Dataset for Mechanical Engineers. URL: https://medium.com/@ahmet17/makina-m%C3%BChendisleri-i%C3%A7in-derin-%C3%B6%C4%9Frenme-3d-printer-veri-setinin-i%CC%87ncelenmesi-6fe1f48e0cdb

[OLSO16a] Olson, R. S.; Moore, J. H.: TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning. In: ML 2016 AutoML Workshop, 2016, DOI: 10.1007/978-3-030-05318-5_8.

[OLSO16b] Olson, R. S.; Bartley, N.; Urbanowicz, R. J.; Moore, J. H.: Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science, 2016, DOI: 10.48550/arXiv.1603.06212.

[OLVE10] Olvera-López, J. A.; Carrasco-Ochoa, J. A.; Martínez-Trinidad, J. F.: A new fast prototype selection method based on clustering. In: Pattern Analysis and Applications, Vol. 13, 2010, No. 2, pp. 131–141, DOI: 10.1007/s10044-008-0142-x.

[ONO69] Ono, M.; Miller, H. P.: Income nonresponses in the current population survey, 1969

[PAAT94] Paatero, P.; Tapper, U.: Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. In: Environmetrics, Vol. 5, 1994, No. 2, pp. 111–126, DOI: 10.1002/env.3170050203.

[PAND14] Pandey, K. K.; Pradhan, N.: An Analytical and Comparative Study of Various Data Preprocessing Method in Data Mining. In: International

Journal of Emerging Technology and Advanced Engineering, Vol. 4, 2014, No. 10, pp. 174–180

[PARR21] Parry, P.: auto_ml. URL: https://github.com/ClimbsRocks/auto_ml [Accessed: 14.07.2021]

[PATI20] Patil, A.: Principal Component Analysis(PCA) - Analytics Vidhya - Medium. URL: https://medium.com/analytics-vidhya/principal-component-analysis-pca-8a0fcba2e30c [Accessed: 28.07.2021]

[PEAR00] Pearson, K.: X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. In: The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, Vol. 50, 1900, No. 302, pp. 157–175, DOI: 10.1080/14786440009463897.

[PEAR01] Pearson, K.: LIII. On lines and planes of closest fit to systems of points in space. In: The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, Vol. 2, 1901, No. 11, pp. 559–572, DOI: 10.1080/14786440109462720.

[PEDR21a] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E.: 1.13. Feature selection — scikit-learn 0.24.2 documentation. URL: https://scikit-learn.org/stable/modules/feature_selection.html [Accessed: 08.08.2021]

[PEDR21b] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E.: 1.1. Linear Models — scikit-learn 0.24.2 documentation. URL: https://scikit-learn.org/stable/modules/linear_model.html [Accessed: 06.08.2021]

[PEDR21c] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E.: 1.4. Support Vector Machines — scikit-learn 0.24.2 documentation. URL: https://scikit-learn.org/stable/modules/svm.html [Accessed: 06.08.2021]

[PEDR21d] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.;

Duchesnay, E.: 1.11. Ensemble methods — scikit-learn 0.24.2 documentation. URL: https://scikit-learn.org/stable/modules/ensemble.html [Accessed: 06.08.2021]

[PEDR21e] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E.: 1.10. Decision Trees — scikit-learn 0.24.2 documentation. URL: https://scikit-learn.org/stable/modules/tree.html [Accessed: 06.08.2021]

[PEDR21f] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E.: sklearn.feature_selection.SequentialFeatureSelector — scikit-learn 0.24.2 documentation. URL: https://scikit-learn.org/0.24/modules/generated/sklearn.feature_selection.SequentialFeatureSelector.html [Accessed: 13.08.2021]

[PEDR21g] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E.: 1.6. Nearest Neighbors — scikit-learn 0.24.2 documentation. URL: https://scikit-learn.org/stable/modules/neighbors.html [Accessed: 06.08.2021]

[PENG08] Peng, Y.; Brazdil, P.; Flach, Peter, A.; Soares, C.: Decision Tree-Based Data Characterization for Meta-Learning, 2008

[PFIR19] Pfirrmann, D.; Voit, M.; Eckstein, M.: Quality Control of a milling process using process data management in the aerospace industry. In: MM Science Journal, 2019, DOI: 10.17973/MMSJ.2019_11_2019052.

[PHAM05] Pham, D. T.; Afify, A. A.: Machine-learning techniques and their applications in manufacturing. In: Proceedings of the Institution of Mechanical Engineers Part B Journal of Engineering Manufacture, Vol. 219, 2005, No. 5, pp. 395–412, DOI: 10.1243/095440505X32274.

[PILL22] Piller, F.: Mass Customization. URL: http://frankpiller.com/mass-customization/

[POPO17] Popova, E.; Rodgers, T. M.; Gong, X.; Cecen, A.; Madison, J. D.; Kalidindi, S. R.: Process-Structure Linkages Using a Data Science Approach: Application to Simulated Additive Manufacturing Data. In: Integrating Materials and Manufacturing Innovation, Vol. 6, 2017, No. 1, pp. 54–68, DOI: 10.1007/s40192-017-0088-1.

[POTD17] Potdar, K.; Pardawala, T. S.; Pai, C. D.: A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers. In: International Journal of Computer Applications, Vol. 175, 2017, No. 4, pp. 7–9, DOI: 10.5120/ijca2017915495.

[POWE02] Power, D. J.: Decision Support Systems: Concepts and Resources for Managers, 67, 2002, ISBN: 9781567204971

[PROK18] Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A.: CatBoost: unbiased boosting with categorical features. In: NIPS'18: Proceedings of the 32nd International Conference on Neural Information Processing Systems, Vol. 31, 2018, DOI: 10.5555/3327757.3327770.

[PULA20] Pulagam, S.: All you need to know about PCA technique in Machine Learning. URL: https://towardsdatascience.com/all-you-need-to-know-about-pca-technique-in-machine-learning-443b0c2be9a1 [Accessed: 28.07.2021]

[PWC17] PWC: Sizing the prize. What's the real value of AI for your business and how can you capitalise? URL: https://www.pwc.com/gx/en/issues/analytics/assets/pwc-ai-analysis-sizing-the-prize-report.pdf

[PWC20] PWC: How AI can enable a Sustainable Future. URL: https://www.pwc.de/de/nachhaltigkeit/how-ai-can-enable-a-sustainable-future.pdf

[PYLE99] Pyle, D.: Data preparation for data mining. San Francisco, Calif.: Morgan Kaufmann, 1999, ISBN: 9781558605299

[QUIN86] Quinlan, J. R.: Induction of decision trees. In: Machine Learning, Vol. 1, 1986, No. 1, pp. 81–106, DOI: 10.1007/BF00116251.

[RABE08] Rabe, M.; Spieckermann, S.; Wenzel, S.: Verifikation und Validierung für die Simulation in Produktion und Logistik: Vorgehensmodelle und Techniken: Springer Science & Business Media, 2008, ISBN: 3540352813

[RAES92] Raes, J.: Inside two commercially available statistical expert systems. In: Statistics and Computing, Vol. 2, 1992, No. 2, pp. 55–62, DOI: 10.1007/BF01889583.

[RAIC05] Raicharoen, T.; Lursinsap, C.: A divide-and-conquer approach to the pairwise opposite class-nearest neighbor (POC-NN) algorithm. In: Pattern Recognition Letters, Vol. 26, 2005, No. 10, pp. 1554–1567, DOI: 10.1016/j.patrec.2005.01.003.

[RAMA01] Raman, V.; Hellerstein, J. M.: Potter's Wheel: An Interactive Data Cleaning System, 2001, pp. 381–390, DOI: 10.5555/645927.672045.

[RASM06] Rasmussen, C. E.; Williams, C.: Gaussian Processes for Machine Learning: Massachusetts Institute of Technology, 2006, ISBN: 026218253X

[REBO04] Rebours, P.: Partitioning filter approach to noise elimination: An empirical study in software quality classification, 2004

[REDM18] Redman, T. C.: If Your Data Is Bad, Your Machine Learning Tools Are Useless. URL: https://hbr.org/2018/04/if-your-data-is-bad-your-machine-learning-tools-are-useless [Accessed: 24.07.2021]

[REDM96] Redman, T. C.: Data Quality for the Information Age. 1st ed., 1996, ISBN: 9780890068830

[REKA17] Rekatsinas, T.; Chu, X.; Ilyas, I. F.; Ré, C.: HoloClean: Holistic Data Repairs with Probabilistic Inference. In: Proceedings of the VLDB Endowment, Vol. 10, 2017, No. 11, pp. 1–13, DOI: 10.14778/3137628.3137631.

[RICH95] Richeldi, M.; Rossotto, M.: Class-driven statistical discretization of continuous attributes (Extended abstract). In: ECML '95: Proceedings of the 8th European Conference on Machine Learning, 1995, pp. 335–338, DOI: 10.5555/645324.649625.

[RINN20] Rinne, J.: Screening datasets for laser welded steel-copper lap joints, 2020, DOI: 10.17632/2S5M3CRBKD.1.

[RIQU03] Riquelme, J. C.; Aguilar-Ruiz, J. S.; Toro, M.: Finding representative patterns with ordered projections. In: Pattern Recognition, Vol. 36, 2003, No. 4, pp. 1009–1018, DOI: 10.1016/S0031-3203(02)00119-X.

[RISS78] Rissanen, J.: Modeling by shortest data description. In: Automatica, Vol. 14, 1978, No. 5, pp. 465–471, DOI: 10.1016/0005-1098(78)90005-5.

[RITT75] Ritter, G.; Woodruff, H.; Lowry, S.; Isenhour, T.: An algorithm for a selective nearest neighbor decision rule (Corresp.). In: IEEE Transactions on Information Theory, Vol. 21, 1975, No. 6, pp. 665–669, DOI: 10.1109/tit.1975.1055464.

[ROIG17] Roiger, R. J.: Data Mining. In: Chapman & Hall / CRC Data Mining and Knowledge Discovery Series, Vol. 2, 2017, DOI: 10.1201/9781315382586.

[ROWE00] Roweis, S. T.; Saul, L. K.: Nonlinear dimensionality reduction by locally linear embedding. In: Science (New York, N.Y.), Vol. 290, 2000, No. 5500, pp. 2323–2326, DOI: 10.1126/science.290.5500.2323.

[ROYB20] Roy, B.: All about Feature Scaling - Towards Data Science. URL: https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35 [Accessed: 07.08.2021]

[RUBI87] Rubin, D. B.: Multiple Imputation for Nonresponse in Surveys: John Wiley & Sons, Inc., 1987, ISBN: 9780470316696, DOI: 10.1002/9780470316696.

[RUME86] Rumelhart, D. E.; Hinton, G. E.; Williams, R. J.: Learning representations by back-propagating errors. In: Nature, Vol. 323, 1986, No. 6088, pp. 533–536, DOI: 10.1038/323533a0.

[RUSS10] Russell, S. J.; Norvig, P.; Davis, E.; Edwards, D.: Artificial intelligence. A modern approach Series: Always learning. Boston, Columbus, Indianapolis, New York, San Francisco, Upper Saddle River, Amsterdam, Cape Town, Dubai, London, Madrid, Milan, Munich, Paris, Montreal, Toronto, Delhi, Mexico City, Sao Paulo, Sydney, Hong Kong, Seoul, Singapore, Taipei, Tokyo: Pearson, 2010, ISBN: 1292153962

[SAAD18] Saadallah, A.; Finkeldey, F.; Morik, K.; Wiederkehr, P.: Stability prediction in milling processes using a simulation-based machine learning approach. In: Procedia CIRP, Vol. 72, 2018, pp. 1493–1498, DOI: 10.1016/j.procir.2018.03.062.

[SAEY07] Saeys, Y.; Inza, I.; Larrañaga, P.: A review of feature selection techniques in bioinformatics. In: Bioinformatics (Oxford, England), Vol. 23, 2007, No. 19, pp. 2507–2517, DOI: 10.1093/bioinformatics/btm344.

[SÁEZ22] Sáez, J. A.; Corchado, E.: ANCES: A novel method to repair attribute noise in classification problems. In: Pattern Recognition, Vol. 121, 2022, DOI: 10.1016/j.patcog.2021.108198.

[SAMI19a] Samiullah, C.: How to Deploy Machine Learning Models. A Guide. URL: https://christophergs.com/machine%20learning/2019/03/17/how-to-deploy-machine-learning-models/

[SAMI19b] Samiullah, C.: Deploying Machine Learning Applications in Shadow Mode. A Guide. URL: https://christophergs.com/machine%20learning/2019/03/30/deploying-machine-learning-applications-in-shadow-mode/

[SAMU59] Samuel, A.: Some Studies in Machine Learning Using the Game of Checkers, Vol. 3, 1959, pp. 210–229, DOI: 10.1147/rd.33.0210.

[SANT17] Santoyo, S.: A Brief Overview of Outlier Detection Techniques - Towards Data Science. URL: https://towardsdatascience.com/a-brief-overview-of-outlier-detection-techniques-1e0b2c19e561 [Accessed: 27.07.2021]

[SARF20] Sarfin, R.: 5 Characteristics of Data Quality. URL: https://www.precisely.com/blog/data-quality/5-characteristics-of-data-quality [Accessed: 01.10.2021]

[SAS22] SAS Enterprise Miner: SEMMA. URL: https://www.sas.com/en_us/software/enterprise-miner.html

[SAXE08] Saxena, A.; Goebel K.: Turbofan Engine Degradation Simulation Data Set. In: NASA Ames Prognostics Data Repository, NASA Ames, Moffett Field, CA., 2008

[SAYY05] Sayyad Shirabad, J.; Menzies, T. J.: The PROMISE Repository of Software Engineering Databases, 2005

[SCHE02] Scheffer, J.: Dealing with missing data. In: Research Letters in the Information and Mathematical Sciences, Vol. 3, 2002, pp. 153–160

[SCHM20] Schmitt, R. H.; Ellerich, M.; Schlegel, P.; Ngo, Q. H.; Emonts, D.; Montavon, B.; Buschmann, D.; Lauther, R.: Datenbasiertes Qualitätsmanagement im Internet of Production. In: Frenz, W. (ed.): Handbuch Industrie 4.0: Recht, Technik, Gesellschaft., 2020, pp. 489–516, DOI: 10.1007/978-3-662-58474-3_25.

[SEGA10] Segata, N.; Blanzieri, E.; Delany, S. J.; Cunningham, P.: Noise reduction for instance-based learning with a local maximal margin approach. In: Journal of Intelligent Information Systems, Vol. 35, 2010, No. 2, pp. 301–331, DOI: 10.1007/s10844-009-0101-z.

[SEME22] Semeion, Research Center of Sciences of Communication: Steel Plates Faults Data Set. URL: https://archive.ics.uci.edu/ml/datasets/Steel+Plates+Faults [Accessed: 10-Feb-22]

[SERB13] Serban, F.; Vanschoren, J.; Kietz, J.-U.; Bernstein, A.: A Survey of Intelligent Assistants for Data Analysis. In: ACM Computing Surveys, Vol. 45, 2013, No. 3, DOI: 10.1145/2480741.2480748.

[SHAN19] Shang, Z.; Zgraggen, E.; Buratti, B.; Kossmann, F.; Eichmann, P.; Chung, Y.; Binnig, C.; Upfal, E.; Kraska, T.: Deomcratizing Data Science through Interactive Curation of ML Pipelines. In: In 2019 International Conference on Management of Data (SIGMOD '19), 2019, DOI: 10.1145/3299869.3319863.

[SIEG12] Siegel, A. F.: Time Series: Understanding Changes over Time. In: Practical Business Statistics, Vol. 6, 2012, pp. 429–464, DOI: 10.1016/B978-0-12-385208-3.00014-6.

[SING20] Singh, D.; Climente-González, H.; Petrovich, M.; Kawakami, E.; Yamada, M.: FsNet: Feature Selection Network on High-dimensional Biological Data, 2020, DOI: 10.48550/arXiv.2001.08322.

[SLIJ20] Slijepcevic, D.; Zeppelzauer, M.; Schwab, C.; Raberger, A.-M.; Breiteneder, C.; Horsak, B.: Input representations and classification strategies for automated human gait analysis. In: Gait & Posture, Vol. 76, 2020, pp. 198–203, DOI: 10.1016/j.gaitpost.2019.10.021.

[SMIT11] Smith M. R.; Martinez T.: Improving classification accuracy by identifying and removing instances that should be misclassified. In: The 2011 International Joint Conference on Neural Networks, 2011, pp. 2690–2697, DOI: 10.1109/IJCNN.2011.6033571.

[SMIT14] Smith, M. R.; Martinez, T.; Giraud-Carrier, C.: An instance level analysis of data complexity. In: Machine Learning, Vol. 95, 2014, No. 2, pp. 225–256, DOI: 10.1007/s10994-013-5422-z.

[SMOL08] Smola, A. and Vishwanathan, S. V. N.: Introduction to Machine Learning: Cambridge: University Press, 2008

[SOKO09] Sokolova, M.; Lapalme, G.: A systematic analysis of performance measures for classification tasks. In: Information Processing & Management, Vol. 45, 2009, No. 4, pp. 427–437, DOI: 10.1016/j.ipm.2009.03.002.

[SORJ05] Sorjamaa, A., Reyhani, N. and Lendasse, A.: Input and structure selection for k-nn approximator. In: Computational Intelligence and Bioinspired Systems, 2005, No. 3512, pp. 985–992, DOI: 10.1007/11494669_121.

[SOTE17] Sotelo, D.: Effect of Feature Standardization on Linear Support Vector Machines. URL: https://towardsdatascience.com/effect-of-feature-standardization-on-linear-support-vector-machines-13213765b812 [Accessed: 29.07.2021]

[SPIT03] Spittle, P.: Gas Turbine Technology. In: Physical Education, Vol. 38, 2003, No. 6, pp. 504–511, DOI: 10.1088/0031-9120/38/6/002.

[SPOK22] SPOK: Entwicklung eines smarten Produktionskonzeptes, Bundesministerium für Wirtschaft und Klimaschutz im Nationalen Programm für Innovation, Technologie und Neue Mobilität unter dem Förderkennzeichen 50RL2050, 2022

[SPRA82] Sprague, R. H.; Carlson, E. D.: Building Effective Decision Support Systems, 1, 1982, ISBN: 978-0130862150

[STEP14] Stephenson, W. R.: Bolts. URL: https://www.openml.org/d/857 [Accessed: 10.02.2022]

[STRE22] Streamlit: Streamlit is a company of tinkerers, engineers, and scientists. URL: https://streamlit.io/

[SU05] Su, C.-T.; Hsu, J.-H.: An extended Chi2 algorithm for discretization of real value attributes. In: IEEE Transactions on Knowledge and Data Engineering, Vol. 17, 2005, No. 3, pp. 437–441, DOI: 10.1109/TKDE.2005.39.

[SUN09] Sun, Y.; Wong, A. K. C.; Kamel, M. S.: CLASSIFICATION OF IMBALANCED DATA: A REVIEW. In: International Journal of Pattern Recognition and Artificial Intelligence, Vol. 23, 2009, No. 4, pp. 687–719, DOI: 10.1142/S0218001409007326.

[SUTH16] Suthaharan, S.: Machine learning models and algorithms for Big Data classification. Thinking with examples for effective learning Series: Integrated series in information systems, 36. 1st ed. New York: Springer Science+Business Media, 2016, ISBN: 9781489976413, DOI: 10.1007/978-1-4899-7641-3.

[SVID20] Svideloc: Target Encoding Vs. One-hot Encoding with Simple Examples. URL: https://medium.com/analytics-vidhya/target-encoding-vs-one-hot-encoding-with-simple-examples-276a7e7b3e64 [Accessed: 28.07.2021]

[SWAL18] Swalin, A.: How to Handle Missing Data - Towards Data Science. URL: https://towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4 [Accessed: 27.07.2021]

[SYAF18] Syafrudin, M.; Alfian, G.; Fitriyani, N. L.; Rhee, J.: Performance Analysis of IoT-Based Sensor, Big Data Processing, and Machine Learning Model for Real-Time Monitoring System in Automotive Manufacturing. In: Sensors (Basel, Switzerland), Vol. 18, 2018, No. 9, pp. 2946–2970, DOI: 10.3390/s18092946.

[TATA13] Tata Consultancy Services Ltd.: The Emerging Big Returns on Big Data, 2013

[TATA14] Tata Consultancy Services Ltd.: Using Big Data for Machine Learning Analytics in Manufacturing, 2014

[TENE00] Tenenbaum, J. B.; Silva, V. de; Langford, J. C.: A global geometric framework for nonlinear dimensionality reduction. In: Science (New York, N.Y.), Vol. 290, 2000, No. 5500, pp. 2319–2323, DOI: 10.1126/science.290.5500.2319.

[THOM15] Thome, R.; Winkelmann, A.: Grundzüge der Wirtschaftsinformatik. Organisation und Informationsentwicklung: Springer, 2015, DOI: 10.1007/978-3-662-46732-9.

[THOR13] Thornton, C.; Hutter, F.; Hoos, H. H.; Leyton-Brown, K.: Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In: Proceedings of the 19th ACM SIGKDD interna-

tional conference on Knowledge discovery and data mining, 2013, pp. 847–855, DOI: 10.1145/2487575.2487629.

[TIBS96] Tibshirani, R.: Regression Shrinkage and Selection Via the Lasso. In: Journal of the Royal Statistical Society: Series B (Methodological), Vol. 58, 1996, No. 1, pp. 267–288, DOI: 10.1111/j.2517-6161.1996.tb02080.x.

[TOME76a] Tomek, I.: Two Modifications of CNN. In: IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-6, 1976, No. 11, pp. 769–772, DOI: 10.1109/TSMC.1976.4309452.

[TOME76b] Tomek, I.: An Experiment with the Edited Nearest-Neighbor Rule. In: IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-6, 1976, No. 6, pp. 448–452, DOI: 10.1109/TSMC.1976.4309523.

[TRUO19] Truong, A.; Walters, A.; Goodsitt, J.; Hines, K.; Bruss, C. B.; Farivar, R.: Towards Automated Machine Learning: Evaluation and Comparison of AutoML Approaches and Tools. In: IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), 2019, pp. 1471–1479, DOI: 10.1109/ICTAI.2019.00209.

[TSUB95] Tsubone, H.; Matsuura, H.; Kimura, K.: Decision support system for production planning - Concept and prototype. In: Decision Support Systems, Vol. 13, 1995, No. 2, pp. 207–215, DOI: 10.1016/0167-9236(93)E0037-E.

[TUKE77] Tukey, J. W.: Exploratory Data Analysis. In: Biometrics, Vol. 33, 1977, No. 4, p.768, DOI: 10.2307/2529486.

[UCLA21] UCLA - University of California Los Angeles: R Library Contrast Coding Systems for categorical variables. URL: https://stats.idre.ucla.edu/r/library/r-library-contrast-coding-systems-for-categorical-variables/#HELMERT [Accessed: 17.08.2021]

[ULRI76] Ulrich, P.; Hill, W.: Wissenschaftliche Grundlagen der Betriebswirtschaftslehre (Teil I). In: WiSt - Wirtschaftswissenschaftliches Studium: Zeitschrift für Ausbildung und Hochschulkontakt, Vol. 7, 1976, pp. 304–309

[ULRI81] Ulrich, P.: Die Betriebswirtschaftslehre als anwendungsorientierte Sozialwissenschaft. In: Geist, M.; Köhler, R. (eds.): Die Führung des Betriebes. Curt Sandig zu seinem 80. Geburtstag gewidmet: Poeschel, 1981, pp. 1–25

[UMIC18] UNIVERSITY OF MICHIGAN SMART LAB: CNC Milling Dataset. Data was extracted using the Rockwell Cloud Collector Agent Elastic software from a CNC milling machine in the System-level Manufacturing and Automation Research Testbed (SMART) at the University of

Michigan. URL: https://www.kaggle.com/shasun/tool-wear-detection-in-cnc-mill [Accessed: 10.02.2022]

[UNIV21] University of Virginia Library Research Data Services + Sciences: Data Types & File Formats. URL: https://data.library.virginia.edu/data-management/plan/format-types/ [Accessed: 06.08.2021]

[VANS13] Vanschoren, J.; van Rijn, J. N.; Bischl, B.; Torgo, L.: OpenML: networked science in machine learning. In: ACM SIGKDD Explorations Newsletter, Vol. 15, 2013, No. 2, pp. 49–60, DOI: 10.1145/2641190.2641198.

[VANS19] Vanschoren, J.: Meta-Learning. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds) Automated Machine Learning. The Springer Series on Challenges in Machine Learning, 2019, pp. 35–61, DOI: 10.1007/978-3-030-05318-5_2.

[VDI-18] VDI-Richtlinien: VDI 3633: „Simulation von Logistik-, Materialfluss und Produktionssystemen", 2018

[VEEN05] Veenman, C. J.; Reinders, M. J. T.: The nearest subclass classifier: a compromise between the nearest mean and nearest neighbor classifier. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 27, 2005, No. 9, pp. 1417–1429, DOI: 10.1109/TPAMI.2005.187.

[VENK14] Venkata Rao, K.; Murthy, B.; Mohan Rao, N.: Prediction of cutting tool wear, surface roughness and vibration of work piece in boring of AISI 316 steel with artificial neural network. In: Measurement, Vol. 51, 2014, No. 1, pp. 63–70, DOI: 10.1016/j.measurement.2014.01.024.

[VEZH07] Vezhnevets, A.; Barinova, O.: Avoiding Boosting Overfitting by Removing Confusing Samples, Vol. 4701, 2007, pp. 430–441, DOI: 10.1007/978-3-540-74958-5_40.

[VILA04] Vilalta, R.; Giraud-Carrier, C.; Brazdil, P.; Soares, C.: Using Meta Learning to Support Data Mining. In: International Journal of Computer Science & Applications, Vol. 1, 2004, No. 1, pp. 31–45

[WANG18] Wang J.; Ma Y.; Zhang Lea: Deep learning for smart manufacturing: Methods and applications. In: Journal of Manufacturing Systems, Vol. 48, 2018, No. C, pp. 144–156, DOI: 10.1016/j.jmsy.2018.01.003.

[WANG96] Wang, R. Y.; Strong, D. M.: Beyond Accuracy: What Data Quality Means to Data Consumers. In: Journal of Management Information Systems, Vol. 12, 1996, No. 4, pp. 5–33, DOI: 10.1080/07421222.1996.11518099.

[WASC18] Waschneck, B.; Reichstaller, A.; Belzner, L.; Altenmüller, T.; Bauernhansl, T.; Knapp, A.; Kyek, A.: Optimization of global production

scheduling with deep reinforcement learning. Procedia CIRP 72 (2018). In: Conference on Manufacturing Systems (CMS), 2018, pp.1264-1269, DOI: 10.1016/j.procir.2018.03.212.

[WEIC19] Weichert, D.; Link, P.; Stoll, A.; Rüping, S.; Ihlenfeldt, S.; Wrobel, S.: A Review of Machine Learning for the Optimization of Production Processes. In: International Journal of Advanced Manufacturing Technologies, 2019, No. 104, pp. 1889–1902, DOI: 10.1007/s00170-019-03988-5.

[WEIN09] Weinberger, K.; Dasgupta, A.; Attenberg, J.; Langford, J.; Smola, A.: Feature Hashing for Large Scale Multitask Learning. In: Proceedings of the 26th Annual International Conference on Machine Learning, 2009, pp. 1113–1120, DOI: 10.1145/1553374.1553516.

[WIDM15] Widmann, M.; Silipo, R.: Seven Techniques for Data Dimensionality Reduction | KNIME. URL: https://www.knime.com/blog/seven-techniques-for-data-dimensionality-reduction [Accessed: 08.08.2021]

[WIJA21] Wijaya, C. Y.: 4 Categorical Encoding Concepts to Know for Data Scientists. URL: https://towardsdatascience.com/4-categorical-encoding-concepts-to-know-for-data-scientists-e144851c6383 [Accessed: 06.03.2022]

[WILL00] Williams, C.; Seeger, M.: Using the Nyström Method to Speed Up Kernel Machines. In: Advances in Neural Information Processing Systems, Vol. 13, 2000, pp. 661–667, DOI: 10.5555/3008751.3008847.

[WILS16] Wilson, T.: Dealing with Outliers (Part 2): Avoid These Common Mistakes. URL: https://www.brooksbell.com/resource/blog/dealing-with-outliers-part-2 [Accessed: 27.07.2021]

[WILS72] Wilson, D. L.: Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. In: IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-2, 1972, No. 3, pp. 408–421, DOI: 10.1109/TSMC.1972.4309137.

[WIND22] Windwärts Energie GmbH: Windenergie. Wie funktioniert eine Windenergieanlage. URL: https://www.windwaerts.de/de/infothek/know-how/funktion-windenergieanlage

[WITT05] Witten, I. H.; Frank, E.: Data Mining. Practical Machine Learning Tools and Techniques. 2nd ed.: Elsevier Inc., 2005

[WOLP97] Wolpert, D.H. and Macready, W.G.: No free lunch theorems for optimization. In: IEEE Transactions on Evolutionary Computation, Vol. 1, 1997, No. 1, pp. 67–82, DOI: 10.1109/4235.585893.

[WONG87a] Wong, A. K.; Chiu, D. K.: Synthesizing statistical knowledge from incomplete mixed-mode data. In: IEEE transactions on pattern analysis

and machine intelligence, Vol. 9, 1987, No. 6, pp. 796–805, DOI: 10.1109/tpami.1987.4767986.

[WONG87b] Wong, A. K.; Chiu, D. K.: An event-covering method for effective probabilistic inference. In: Pattern Recognition. In: Pattern Recognition, Vol. 20, 1987, No. 2, pp. 245–255, DOI: 10.1016/0031-3203(87)90058-6.

[WU03] Wu, G.; Chang, E. Y.: Class-boundary alignment for imbalanced dataset learning. In: ICML 2003 Workshop on Learning from Imbalanced Data Sets, 2003, pp. 49–56

[WU18] Wu, D.; Jennings, C.; Terpenny, J.; Kumara, S.; Gao, R. X.: Cloud-Based Parallel Machine Learning for Tool Wear Prediction. In: Journal of Manufacturing Science and Engineering, Vol. 140, 2018, No. 4, DOI: 10.1115/1.4038002.

[WU96] Wu, X.: A Bayesian Discretizer for Real-Valued Attributes. In: The Computer Journal, Vol. 39, 1996, No. 8, pp. 688–691, DOI: 10.1093/comjnl/39.8.688.

[WUES16] Wuest, T.; Weimer, D.; Irgens, C.; Thoben, K.-D.: Machine learning in manufacturing: advantages, challenges, and applications. In: Production & Manufacturing Research, Vol. 4, 2016, No. 1, pp. 23–45, DOI: 10.1080/21693277.2016.1192517.

[WUWE19] Wu, W.; Ye, J.; Wang, Q.; Luo, J.; Xu, S.: CT-Based Radiomics Signature for the Preoperative Discrimination Between Head and Neck Squamous Cell Carcinoma Grades. In: Frontiers in Oncology, Vol. 9, 2019, pp. 1–9, DOI: 10.3389/fonc.2019.00821.

[YAN20] Yan, C.; He, Y.: Auto-Suggest: Learning-to-Recommend Data Preparation Steps Using Data Science Notebooks. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), 2020, pp. 1–16, DOI: 10.1145/3318464.

[YANG01] Yang, Y.; Webb, G. I.: Proportional k-Interval Discretization for Naive-Bayes Classifiers. In: De Raedt, L., Flach, P. (eds) Machine Learning: ECML 2001. ECML 2001. Lecture Notes in Computer Science, Vol. 2167, 2001, pp. 564–575, DOI: 10.1007/3-540-44795-4_48.

[YANG03] Yang, Y.; Webb, G. I.: On Why Discretization Works for Naive-Bayes Classifiers. In: Gedeon, T.(.D., Fung, L.C.C. (eds) AI 2003: Advances in Artificial Intelligence. AI 2003. Lecture Notes in Computer Science, Vol. 2903, 2003, pp. 440–452, DOI: 10.1007/978-3-540-24581-0_37.

[YANG19] Yang, X.-S.: Introduction to Algorithms for Data Mining and Machine Learning: Academic Press, 2019, ISBN: 9780128172179

[YEO00] Yeo, I.-K.: A new family of power transformations to improve normality or symmetry. In: Biometrika, Vol. 87, 2000, No. 4, pp. 954–959, DOI: 10.1093/biomet/87.4.954.

[YIN15] Yin, S.; Kaynak, O.: Big Data for Modern Industry: Challenges and Trends. In: Proceedings of the IEEE, Vol. 103, 2015, No. 2, pp. 143–146, DOI: 10.1109/JPROC.2015.2388958.

[YIUT19] Yiu, T.: The Curse of Dimensionality - Towards Data Science. URL: https://towardsdatascience.com/the-curse-of-dimensionality-50dc6e49aa1e [Accessed: 29.07.2021]

[YURC21] Yurchak, R.; Mathieu, T.; Aridas, C.; Joswig, J.-O.: Robust algorithms for Regression, Classification and Clustering. URL: https://github.com/scikit-learn-contrib/scikit-learn-extra/blob/eb129c4e1e5775c35eb4314e9b98b06dbb5160ea/doc/modules/robust.rst [Accessed: 22.07.2021]

[ZAGA21] Zagatti, F. R.: Data Preparation Pipeline Recommendation via Meta-Learning. UNIVERSIDADE FEDERAL DE SÃO CARLOS, 2021

[ZAKO11] Zakova, M.; Kremen, P.; Zelezny, F.; Lavrac, N.: Automating Knowledge Discovery Workflow Composition Through Ontology-Based Planning. In: IEEE Transactions on Automation Science and Engineering, 2011, pp. 253–264, DOI: 10.1109/TASE.2010.2070838.

[ZHAN21] Zhang, H.; Zhang, X.; Song, M.: Deploying AI for New Product Development Success. In: Research-Technology Management, 2021, No. 64, pp. 50–57, DOI: 10.1080/08956308.2021.1942646.

[ZHAO18] Zhao, Y.; Hryniewicki, M. K.: XGBOD: Improving Supervised Outlier Detection with Unsupervised Representation Learning, 2018

[ZHAO19] Zhao, Y.; Nasrullah, Z.; Hryniewicki, M. K.; Li, Z.: LSCP: Locally Selective Combination in Parallel Outlier Ensembles. In: Proceedings of the 2019 SIAM 2019, 2019, pp. 585–593, DOI: 10.1137/1.9781611975673.66.

[ZHAO21] Zhao, Y.; Hu, X.; Cheng, C.; Wang, C.; Wan, C.; Wang, W.; Yang, J.; Bai, H.; Li, Z.; Xiao, C.; Wang, Y.; Qiao, Z.; Sun, J.; Akoglu, L.: SUOD: Accelerating Large-Scale Unsupervised Heterogeneous Outlier Detection. In: Proceedings of Machine Learning and Systems, Vol. 3, 2021, pp. 463–478

[ZHAX19] Zhang, X.; Ran, J.; Mi, J.: An Intrusion Detection System Based on Convolutional Neural Network for Imbalanced Network Traffic. In: IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT), 2019, pp. 456–460, DOI: 10.1109/ICCSNT47585.2019.8962490.

[ZHEN18b] Zheng, P.; wang, H.; Sang, Z.; Zhong, R. Y.; Liu, Y.; Liu, C.; Mubarok, K.; Yu, S.; Xu, X.: Smart manufacturing systems for Industry 4.0: Conceptual framework, scenarios, and future perspectives. In: Frontiers of Mechanical Engineering, Vol. 13, 2018, No. 2, pp. 137–150, DOI: 10.1007/s11465-018-0499-5.

[ZHOU08] Zhou, F.; Yang, B.; Li, L.; Chen, Z.: Overview of the New Types of Intelligent Decision Support System. In: IEEE 3rd International Conference on Innovative Computing Information and Control, 2008, DOI: 10.1109/ICICIC.2008.412.

[ZHU03] Zhu, X.; Wu, X.; Chen, Q.: Eliminating Class Noise in Large Datasets. In: Proceedings of the 20th International Conference on Machine Learning (ICML-03), 2003, pp. 920–927, DOI: 10.5555/3041838.3041954.

[ZHU06] Zhu, X.; Wu, X.: Class Noise Handling for Effective Cost-Sensitive Learning by Cost-Guided Iterative Classification Filtering. In: IEEE Transactions on Knowledge and Data Engineering, Vol. 18, 2006, No. 10, pp. 1435–1440, DOI: 10.1109/tkde.2006.155.

[ZHUP14] Zhu, P.; Zhan, X.; Qiu, W.: Efficient k-Nearest Neighbors Search in High Dimensions Using MapReduce. In: IEEE International Conference, 2014, pp. 23–30, DOI: 10.1109/BDCloud.2015.51.

[ZIGH98] Zighed, D. A.; Rabaséda, S.; Rakotomalala, R.: FUSINTER: A Method for Discretization of Continuous Attributes. In: International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, Vol. 6, 1998, No. 3, pp. 307–326, DOI: 10.1142/S0218488598000264.

[ZIMM20] Zimmer, L.; Lindauer, M.; Hutter, F.: Auto-PyTorch Tabular: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL, 2020, DOI: 10.48550/arXiv.2006.13799.

[ZOU05] Zou, H.; Hastie, T.: Regularization and variable selection via the elastic net. In: Journal of the Royal Statistical Society: Series B (Statistical Methodology), Vol. 67, 2005, No. 2, pp. 301–320, DOI: 10.1111/j.1467-9868.2005.00503.x.

Student Theses

The results presented in this thesis are partly based on findings obtained in the following student theses, which were co-supervised by the author:

Julian Breitling	*Performance Evaluation of Machine Learning-Algorithms for Quality Improvement in High-Speed-Milling Process*. Master Thesis. Faculty of Engineering. Albstadt-Sigmaringen University. 2019
Tristan Johannes Beuting	*Entwicklung eines standardisierten Datenaufbereitungsverfahrens zur Machine Learning-basierten Vorhersage des Werkzeugverschleißes von CNC-Fräsprozessen.* Master Thesis. Chair of Production Metrology and Quality. RWTH Aachen University. 2019
Johannes Mohren	*Structuring and Benchmarking of Data Preprocessing Methods for Machine Learning Applications in Production*. Bachelor Thesis. Chair of Production Metrology and Quality. RWTH Aachen University. 2020
Fabian Bussieweke	*Konzeptentwicklung für die Automatisierung der Datenvorverarbeitung für Machine Learning-Anwendungen in der Produktion*. Bachelor Thesis. Chair of Production Metrology and Quality. RWTH Aachen University. 2020
Linda Weikl	*Machine Learning Based Product Quality Prediction in Blisk Manufacturing*. Master Thesis. Chair of Production Metrology and Quality. RWTH Aachen University. 2020
Johannes Beneke	*Benchmarking of Data Preprocessing Pipelines for Machine Learning Applications in Production*. Master Thesis. Chair of Production Metrology and Quality. RWTH Aachen University. 2021
Petros Tsialis	*Entwicklung eines Intelligent Decision Support Systems zur Auswahl von geeigneten Data Pre-Processing Pipelines*. Bachelor Thesis. Wirtschaftsinformatik. Hochschule für Technik Stuttgart. 2022

Rohan Balaji — *Machine learning-based tool path deviation in Blisk machining.* Master Thesis. Chair of Production Metrology and Quality. RWTH Aachen University. 2022

Charlie Albietz — *Multi-MetaRank: Preprocessing Pipeline Rank Prediction Using Multivariate Regression Meta-Learning and Diverse Meta-Target Selection*. Master Thesis. Faculty of Mathematics and Natural Sciences, Artificial Intelligence. University of Groningen, Netherlands. 2023

VI Annex

A.1 Categories and Methods of Data Cleaning

In the following, the different DPP categories of data cleaning including its methods are described and listed. First, DPP methods of missing value handling are presented and described. Subsequently, outlier handling and noisy data handling methods are discussed.

I. Missing value handling

The long list of 31 DPP methods for missing value handling can be found in Figure A-1. In the following, the missing value handling methods are described following the classification of ignoring, deleting and imputing missing values. Methods, whose origin was found in literature are also referenced with the primary source.

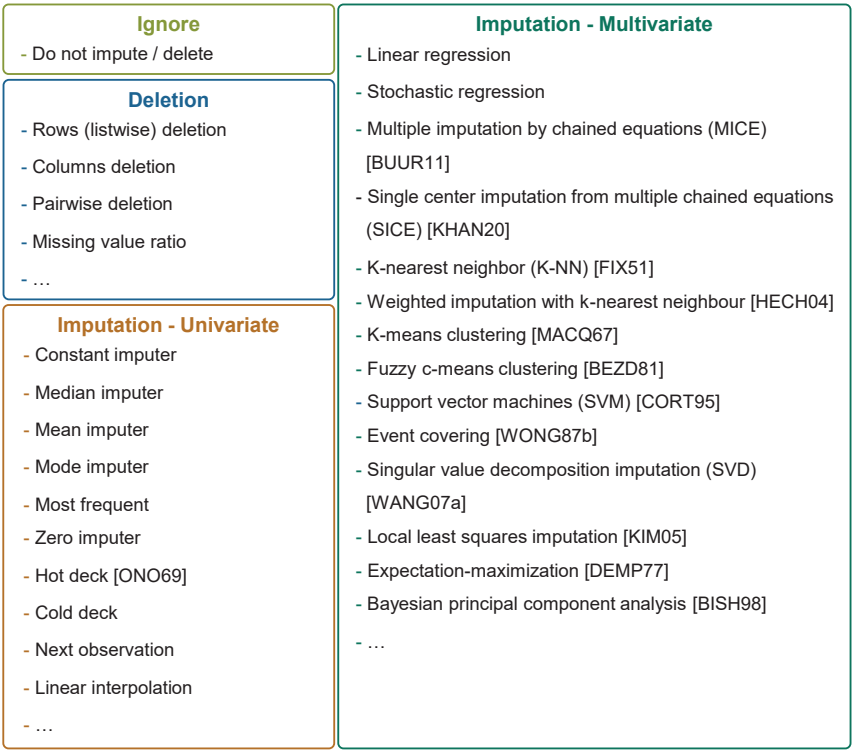


Figure A-1: DPP methods for the DPP category "missing value handling"

Ignore & deletion

A first approach of MV handling is ignoring, which leads to an unbiased modelling. Ignoring can be applied for all types of missing data. [GARC15, p. 63]. For MCAR data, instances and columns can be deleted without influencing subsequent modelling [GARC15, p. 63]. Removing rows is also called listwise deletion. Besides rows, columns can also be deleted. In pairwise deletion, only those columns are taken into account that contain data in all input attributes given the respective target variable. A fourth technique represents the *missing value ratio*. If a certain proportion of missing values exceeds a previously defined threshold, columns or rows are removed. For instance, if 3,000 of 5,000 instances are missing values and a threshold lies at 50 %, then the feature spindle speed is removed from the data set. In conclusion, discarding data points with missing values can have a negative effect in ML applications through losing relevant information [GARC15, p. 63], [SWAL18].

Univariate imputation

Univariate imputation approaches represent fast methods that work well on smaller data sets [GARC15, p. 64]. Missing values are replaced by statistical location parameters of the corresponding feature or manually defined parameters. Examples are *mean, median, mode, constant, zero, hot deck, cold deck* or *MostFrequentImputer*. [GARC15, p. 64], [SWAL18], [ONO69] These approaches do not consider relations between different features being inappropriate for addressing MNAR problems. The application of statistical location parameters is suitable for continuous data. Both numerical and categorical data can be imputed by the *last observation* and *next observation* [SWAL18]. For purely categorical features, the *MostFrequentImputer* value can be used [BADR19]. If time series data is available, missing values can be interpolated linearly or cubically, for instance. [FANC21, pp. 2-4]

Multivariate Imputation

Multivariate imputation approaches cover more than one attribute for replacing missing values. Based on a set of inputs, the missing value is predicted. Since it takes more than one variable into account, these approaches address MAR values, however, require more computations compared to ignoring, deletion and univariate technqiues. The *linear* or *stochastic regression* are prominent examples. Given the features spindle speed, temperatures and sound pressure level, a missing value of an instance can be determined. *K-nearest neighbors* (*K-NN*) represent an ML-based approach that predict MVs for a new instance by taking into account the whole data set [FIX51]. Therefore, *K-NN* addresses both types of MAR and MCAR. New instances are calculated based on k most similar instances, i. e., neighbors, through distance metrics. *K-NN* are especially applicable for small data sets, however, in the presence of high dimensional data,

a performance decrease is to be expected [HAN12, p. 578]. In addition, data needs to be scaled prior to the execution of *K-NN*, while model's outcome is not interpretable. Another ML based example is represented by *K-means clustering* [MACQ67].

Multiple imputation as another more sophisticated approach for handling missing value was proposed by RUBIN [RUBI87]. A realization of multiple imputation is represented by *multiple imputation based on chained equations* (MICE). [BUUR11] A further developed variant is *single center imputation from multiple chained equations* (SICE) [KHAN20]. Further examples for multivariate imputation are *weighted K-NN* [HECH04], *fuzzy k means clustering* [BEZD81], *SVM* [CORT95], *event covering* [WONG87b], *singular value decomposition* [BRAN02], *local least squares* [KIM05], *expectation maximization* [DEMP77], *Bayesian PCA* [BISH98].

II. Outlier handling

Figure A-2 shows a long list of 17 DPP methods for outlier detection. For methods whose origin was found in the literature, reference is provided to the primary source.

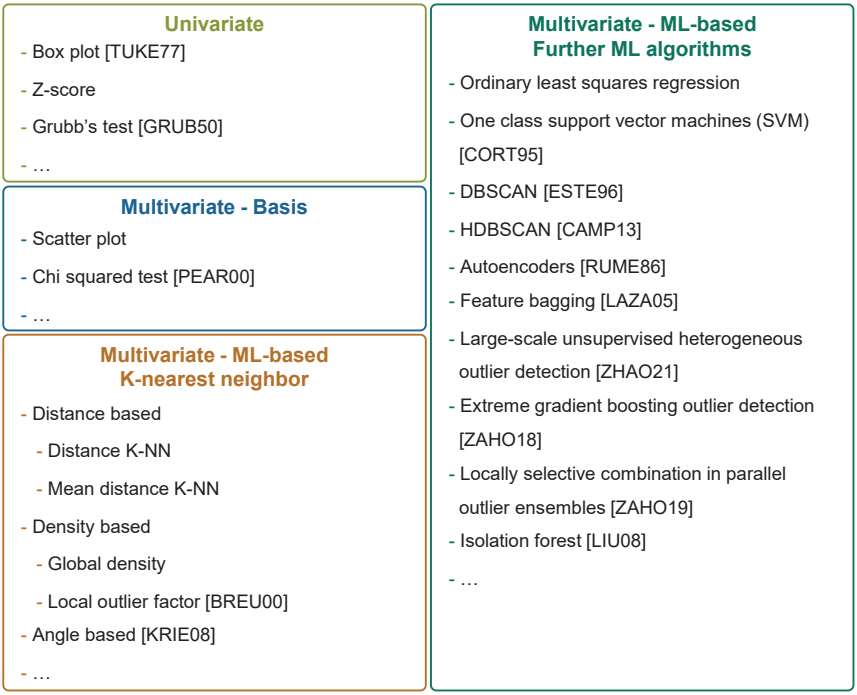


Figure A-2: DPP methods for the DPP category "outlier handling"

Univariate

Univariate methods can detect global outliers. Common techniques are *box plots*, *z score*, and *Grubb's test* [HAN12, pp. 554-555], [GRUB50]. *Box plots* show the data through a box revealing the interquartile range (IQR). Whiskers show the 1.5 times distance of the IQR. All data points that lie outside these whiskers are outliers. [TUKE77] The results of the *box plots* need to be analyzed relatively, since a change in the data distribution also changes the thresholds for outliers. Another outlier detection technique is represented by *z score*, which is equal to a *StandardScaler* in terms of functionality. Given a certain threshold value, outliers are detected if the data point exceeds this value. A typical threshold value is three that needs to be set manually. [FROS19]

Multivariate (basis)

Methods for identifying global outliers in a multidimensional space are *scatter plots* or *chi squared tests*. *Scatter plots* are capable of visualizing two-dimensional data. Since outliers are identified by humans, this approach is error-prone due to human subjectivity [WITT05, p. 313]. Another multivariate example are *Chi-squared tests* [PEAR00].

Multivariate (K-NN / ML based)

When identifying contextual outliers, distance- or density-wise *K-NNs* are used by applying different distance measures. If distances in the proximity of k-1 neighbors are within a given threshold value and neighbor k reveals a higher distance in comparison with its k-1 neighbors, the data point represents an outlier. Besides the distance-based *K-NN*, the density-based *local outlier factor* assigns outlier scores to every instance. Given another density, i. e., distance score, the data point is seen as outlier [HAN12, p. 566], [BREU00]. *Local outlier factor* performs well for data sets exhibiting versatile areas of different density, however, it requires high computational power and shows non-transparency in high dimensions [BREU00, p. 11]. Another example for *K-NN* outlier detection approaches is the angle based technique being especially applied in high dimensional data [KRIE08].

By means of ML, outliers can be identified. Examples are *linear regression*, *SVMs* [CORT95], *DBSCAN* [ESTE96], *HDBSCAN* [CAMP13] or *autoencoders* [RUME86]. *Linear regression*, which is based on ordinary least squares, are prone to outliers thus being suitable for its detection [GARC15, p. 3]. Further examples range from *feature bagging* [LAZA05], *large-scale unsupervised heterogeneous outlier detection* [ZHAO21], *extreme gradient boosting outlier detection* [ZHAO18], *locally selective combination in parallel outlier ensembles* [ZHAO19] or *isolation forest* [LIU08a].

In practice, global outliers can be identified through simple methods that are commonly used. Hereto, univariate, or multivariate methods can be used. Detecting contextual outliers require more complex methods, resulting in methods that stem from ML. As mentioned, collective outliers are very hard to identify in practice. The treatment of outliers can comprise ignoring, deleting, and imputing. Analogous to missing values, removing data points result in an information deprivation, while imputing retains the data set's shape. [GARC15, pp. 63-65], [SWAL18] In general, every deletion and imputation method presented above is applicable to outliers. ABDALLAH ET AL. describe outliers as their own category equal to the treatment of missing values [ABDA17]. Another way of outlier handling is smoothing outliers through the application of the *Minkowski error*. The *Minkowski error* is a loss index being insensitive to outliers, which reduces the impact of outliers on the model. By calculating mean squared error to every error instance, the *Minkowski parameter* reduces the contribution of outliers to the total error through reducing the exponent values from 2.0 to 1.5 or similar. [HANS87]

III. Noisy data handling

In Figure A-3, a long list of 16 DPP methods for noisy data handling is provided. Outlier detection methods are described following the classification of instance, attribute, and class noise. For further reading, the author refers to [GARC15, pp. 107-146]

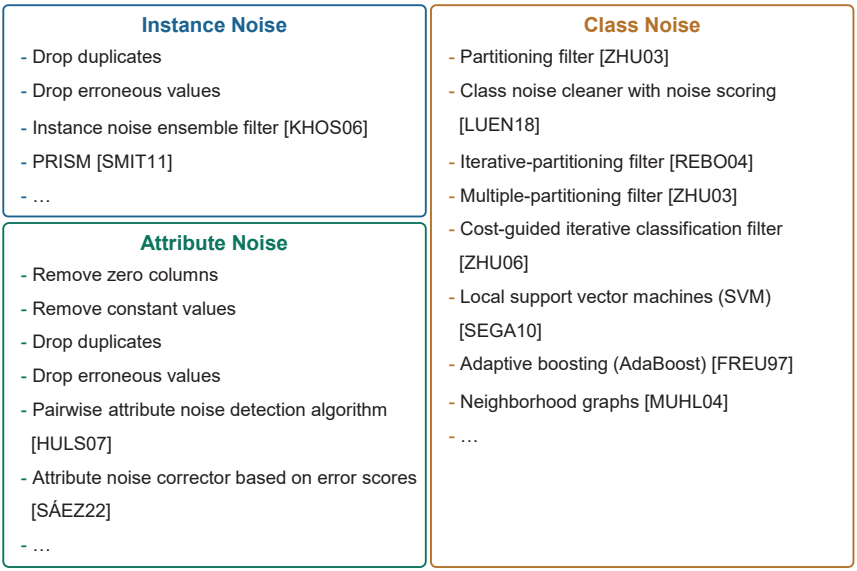


Figure A-3: DPP methods for the DPP category "noisy data handling"

For methods, whose origin was found in literature, reference is made to the primary source. On a broader view, noise is everything but the true signal. Therefore, outliers and MVs can also be assigned to noise [SANT17]. Further forms of noise are duplicates, inconsistent values, unimportant or highly correlated features as well as very volatile data [BROW20b].

Instance noise

Instance noise is addressed through dropping duplicates and erroneous values. Furthermore, *instance noise ensemble filter* [KHOS06] and *PRISM*, i. e., preprocessing instances that should be misclassified [SMIT11], represent examples for handling instance noise.

Attribute noise

The presence of duplicates may result in a deterioration in ML model performance in real word in case duplicates are existing in both training and test data sets [BROW20b]. *Data deduplication* aims at removing repetitive and redundant information, which serves as data reduction approach, while requiring lower computing time during analysis. *Removing constant features* enhance analysis' efficiency. [WITT05, p. 397] In addition, *pairwise attribute noise detection algorithm* [HULS07] and *attribute noise corrector based on error scores* [SÁEZ22] are two additional examples of DPP methods for attribute noise handling.

Class noise

Multiple noise filters have been introduced to address class noise. According to Figure A-3, examples range from *partitioning filter* [ZHU03] over the *iterative-partitioning filter* [REBO04] or the *class noise cleaner with noise scoring* [LUEN18] to *multiple-partitioning filter* [ZHU03], and *cost-guided iterative classification filter* [ZHU06]. Besides these examples, the *local SVM* [SEGA10] is used as well as *adaptive boosting* [FREU97] and *neighborhood graphs* [MUHL04].

A.2 Categories and Methods of Data Transformation

Data transformation contains data encoding, feature scaling, handling skewness as well as discretization. Corresponding DPP methods are discussed according to the DPP categories in the following.

I. Data encoding

Data encoding converts categorical into numerical features that can be classified into classic, Bayesian and contrast encoders, which can be taken Figure A-4. Common techniques applicable in production are described in the following.

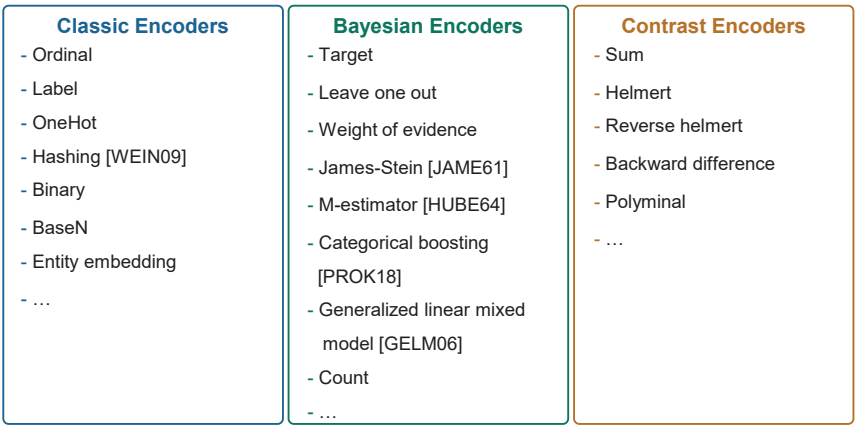


Figure A-4: DPP methods for the DPP category "data encoding"

Classic encoders

OneHotEncoders are also called *DummyEncoders* that encode categorical features based on the cardinality of the feature. While *OneHotEncoder* are applied in case of nominal data, the *OrdinalEncoder* is used if an order of the categories is present. Figure A-5 shows an example, where the categorical feature 'tool wear' containing the states 'low', 'medium' and 'severe' is transformed and encoded into three new features [BROW19a]. *OneHotEncoding* may result in a very high dimensionality and sparse data set depending on the feature's cardinality [SVID20]. Therefore, encoders like *hashing* [WEIN09, p. 1], *binary* and *BaseN* and *entity embedding* encoders have been developed.

Initial Data Set		Label Encoded		One Hot Encoded			
	Tool wear		Tool wear		Tool wear _ low	Tool wear _ medium	Tool wear _ high
0	low	0	1	0	1	0	0
1	medium	1	2	1	0	1	0
2	high	2	3	2	0	0	1

Figure A-5: Label and one hot encoding (adapted from [BROW19a])

Bayesian encoders

Bayesian encoders consider the target variable when encoding categorical columns [MICC01, p. 28]. Equation (A-1) shows the calculation for the encoded variable. Given x_i as categorical variable, then the encoded numeric value, also probability, for the categorical variable p_{TE_i} results from the quotient of the number of matching assignments of the attribute in the target $total\ true(x_i)$ to the total number of occurring entries of the categorical variable in the target $total(x_i)$.

$$p_{TE_i} = \frac{total\ true(x_i)}{total(x_i)} \tag{A-1}$$

Figure A-6 provides an example for target encoding. The categorical variable tool wear comprises *medium* three times. However, only two out of three entries are actually assigned as *medium* tool wear, which is represented through the quality. For this reason, the encoded tool wear receives a probability of $p_{TE_{medium}}$ = 0.67 for these instances. [BUIH20], [SVID20]

	Tool Wear	Quality	Wear _ encoded
0	High	0	0.50
1	Medium	1	0.67
2	Medium	1	0.67
3	High	1	0.50
4	Medium	0	0.67
5	Low	1	1.00

	Target		
Trend	0	1	Probability
Low	0	1	1.00
Medium	1	2	0.67
High	1	1	0.50

Figure A-6: Target encoding (adapted from [SVID20])

TargetEncoders perform for attributes with high cardinality [HALE18], while the fact that taking the target variable into calculation of probabilities may result in overfitting

[LARI20]. The effect of overfitting can be reduced by regularization, introduction of *random noise* or *double validation* [GHOJ19]. *Leave one out encoder* represent another type of Bayesian encoder closely related to the *TargetEncoder* [WIJA21]. Further Bayesian encoders are *weight of evidence*, *James-Stein* [JAME61], *M-estimator* [HUBE64], *categorical boosting encoder* [PROK18], and *generalized linear mixed model* [GELM06].

Contrast encoders

Contrast encoders apply contrasts for encoding. A contrast is a linear combination of attributes whose coefficients or weights add up to zero [CARE03, pp. 10-11]. Contrasts compare interrelated categories, e. g., relationships between medium and high tool wear. The number of new dimensions is equal to the amount of categories-1. Examples range from s*um encoders,* over *Helmert*, to *backward difference*. A further example can be seen in *polynomial encoders*. Contrast encoders are new methods, not well known or popular and, thus, also not applied in production context. [BUIH20], [UCLA21]

II. Feature scaling

The *StandardScaler* scales the data according to Equation (A-2) [BHAN20] that centers the data with respect to the mean. The scaled value X_{new} is the subtraction of the unscaled value X and the mean value μ of the attribute divided by the standard deviation of the attribute σ. [GÉRO18, pp. 66-67]

$$X_{new} = \frac{X - \mu}{\sigma} \tag{A-2}$$

In *normalization*, values of each feature are transformed into values between 0 and 1. The *MinMaxScaler* scales features according to Equation (A-3). [BHAN20] The scaled value X_{new} results from the subtraction of the unscaled value X and the minimum of the column X_{min}divided by the subtraction of the maximum X_{max} and minimum value of the column X_{min}.

$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{A-3}$$

One drawback of the *MinMaxScaler* is its sensitivity to outliers. When scaling of data points including outliers is conducted with the *MinMaxScaler*, all other values except the outlier will be found in a very small range. Given 1,000 data points that are scaled from which 999 of the points exhibit values in the range of 10 and 100 and only one outlier consists a value of 1,000, using Equation (A-3) leads to the situation that

all 999 data points will be scaled to a small range between 0.00 and 0.09. Only the outlier will be scaled to the value of 1. [CODE21] To address this issue, the *Robust-Scaler* takes into account both the median and IQR of the distribution. The *Robust-Scaler* is not bounded. Lastly, the *MaximumAbsoluteScaler* transforms data by scaling to the maximum absolute value. These four different feature scaling techniques can be found in Figure A-7.

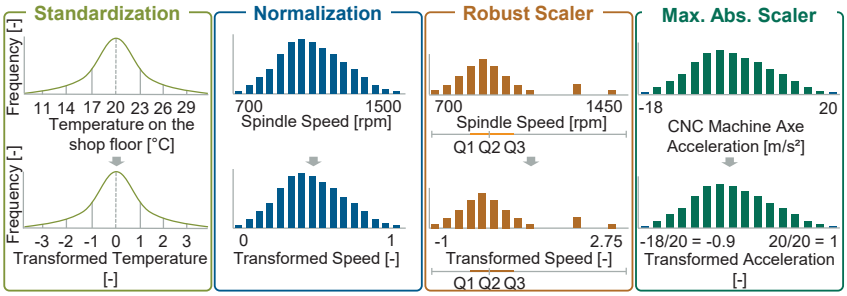


Figure A-7: Illustration of four different feature scaling techniques

III. Handling skewness

Skewness is the measure of asymmetry of a probability distribution. Skewed data can exhibit any common distribution shape. Since ML algorithms prefer data in normal distribution, so called power transformers are used to remove skewness [GARC15, p. 54]. In case data is skewed, applicable power transformers are *square root*, *cube root* and *log transform* [HAN12, p. 106]. In order to transform skewed data independently from the type of skewness, power transformers *Box-Cox* [BOX64] and *Yeo-Johnson* [YEO00] have been developed. Figure A-8 shows the different power transformers that can be implemented given the type of skewness. In case of *Box-Cox*, values are raised to the power of the hyperparameter λ.

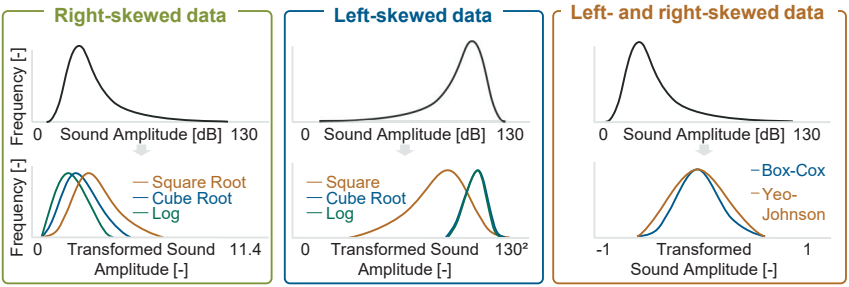


Figure A-8: Power transformers dependent on data skewness at hand

Based on λ, a normal distribution can be achieved. For *Box-Cox transformations*, typical values are between λ = -2 and λ = 2. The downside of *Box-Cox* is the limited applicability to positive data, which is overcome by *Yeo-Johnson*. *Yeo-Johnson* represents an extension of *Box-Cox*, since it is equal to *Box-Cox* for positive values. The *Yeo-Johnson* transformation applies the parameter to provide a distribution which is closest to a normal distribution [BROW20e]. The power parameter is set in the range from λ = -1 to λ = 1 [KUHN18, p. 32].

IV. Data discretization

The long list of 17 DPP methods for data discretization can be found in Figure A-9. Data discretization can be classified into binning, statistical and information methods. For methods, whose origin was found in literature is also referenced with the primary source. For further reading, the author refers to [GARC15, pp. 245-284].

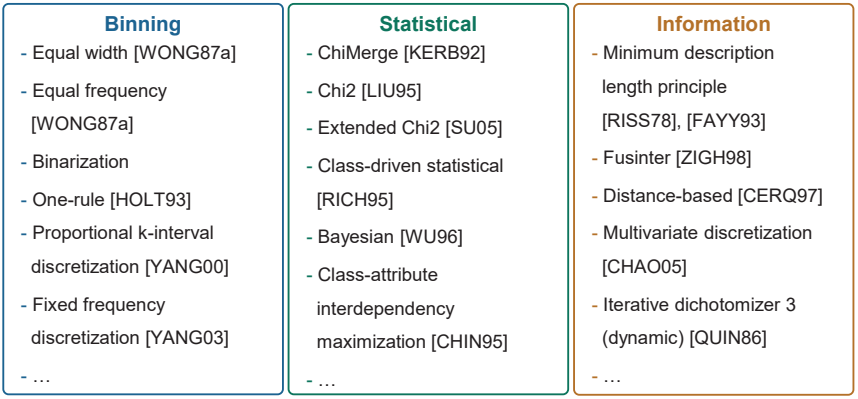


Figure A-9: DPP methods for the DPP category "data discretization" in simplified form from [GARC15, pp. 254-258]

Regarding binning, *equal width* and *equal frequency* are commonly applied [WONG87a]. Further examples are *one-rule* [HOLT93], *proportional k-interval discretization* [YANG01], and *fixed frequency discretization* [YANG03]. For statistical-based discretization, *ChiMerge* [KERB92], *Chi2* [LIU95], *extended Chi2* [SU05], *class-driven statistical* [RICH95], Bayesian [WU96] or *class-attribute interdependency maximization* [CHIN95] are used. In case of information-based data discretization, the methods *minimum description length principle* [RISS78], [FAYY93], *Fusinter* [ZIGH98], *distance-based* [CERQ97], *multivariate* [CHAO05], or *iterative dichotomizer 3 (dynamic)* [QUIN86] are used.

A.3 Categories and Methods of Data Reduction

Data reduction consists of the categories of dimensionality reduction, feature selection, and instance selection. A list of DPP methods according to the DPP categories are classified and described in the following subsections.

I. Dimensionality reduction

In case of dimensionality reduction, entirely new attributes are generated based on existing ones. The idea is to reduce the dimensional space when creating new features. 19 DPP methods for dimensionality reduction are presented, which can be classified into components-, projections-, and ML-based (see Figure A-10).

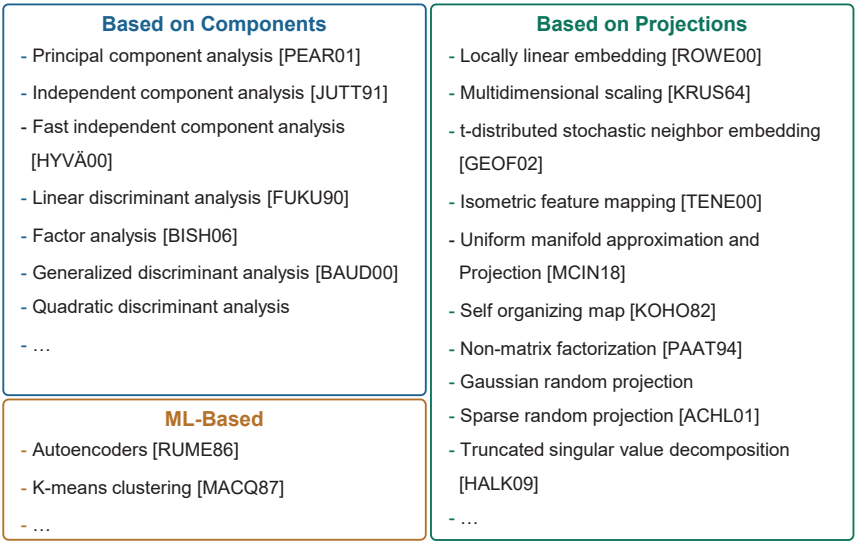


Figure A-10: DPP methods for the DPP category "dimensionality reduction"

Components-based

Principal component analysis (*PCA*) represents a popular technique for dimensionality reduction [PEAR01]. Principal components (PC) that represent new features in the lower dimensional space, aim at describing the most possible variance in the data set. In the first step, the data needs to be centered through scaling, which is achieved through standardization. The scaling considers that features that have different ranges may contribute equally to the performance of the ML model. [PULA20], [PATI20] Afterwards, a line is fitted, which maximizes the sum of the squared distances from projected points to the origin (PC1). The second PC is perpendicular to PC1 that also maximizes the sum of squared distances from projected points to the origin, thus, being

uncorrelated to PC1. In conclusion, the number of PCs corresponds the number of dimensions. The first principal components contain most variance of the initial data set, while the last PCs exhibit smaller variances. [JOLL02, pp.1-5] To reduce the dimensionality of the data set, a hyperparameter is applied that explains how much variance should be retained in the data set. Those PCs with high variance are taken until a variance threshold, i. e., the hyperparameter, is reached. Usual numbers are 95 %. [GARC15, pp. 149-151]. For visualization purposes, scree plots are used. Since new attributes are built, PCs are non-interpretable, i. e., the headers of the attributes disappear when applying *PCA*. [JOLL02, p. 269], [JOLL16, p. 11]. The whole concept of *PCA* is shown in Figure A-11.

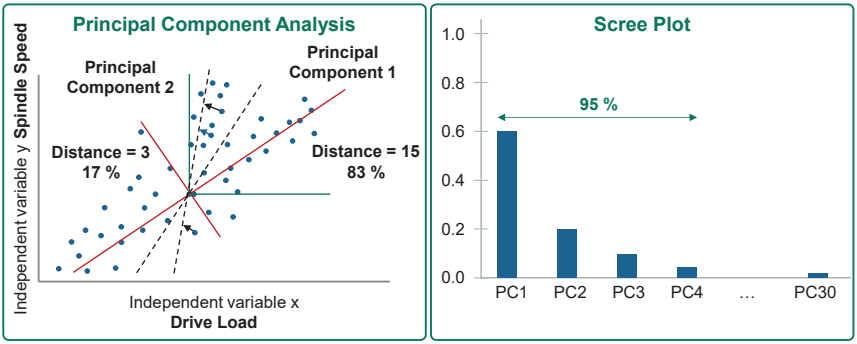


Figure A-11: Concept of principal component analysis (PCA)

Besides *PCA*, *linear discriminant analysis* (LDA) [FUKU90] concentrates on maximizing the separability across labelled categories [KAPR20]. As a first step, the separability between different classes is calculated by maximizing distances between means of classes, called “between-class variance”. Then, the distance between mean and sample of each class is computed though minimizing variation within each category, called “within-class variance”. The number of features is number of classes -1, while it is assumed that data is Gaussian distributed that requires the application of a power transformer beforehand. LDA is highly sensitive to imbalanced data because of concentrating on labelled categories. [KATA20]

While *PCA* focuses on variances, each individual new feature is mutually statistical independent, when *independent component analysis* is applied. [JUTT91]. A further development represents *fast independent component analysis* [HYVÄ00]. *Factor analysis* comprises the distinction of many variables into groups through extracting maximum common variance from all variables and putting them into common scores. Variables in one group of high correlations with each other [BISH06]. A further example based on components is *generalized discriminant analysis* [BAUD00].

Projections-based

A popular technique for projection-based dimensionality reduction represents *t-distributed stochastic neighbor embedding* (t-SNE) [HINT02]. Starting with an original data set, the first step is to compute pairwise similarity for all data points to have a matrix of similarity scores in high dimension [MAAT08], [HINT02]. For that, distances between data points are measured, a normal curve is plotted, which is centered on the point of interest. By using the t-distribution, the similarity of data points can be calculated. The t-distribution is used because the clusters would all clump up in the middle. For all points, similarity scores are calculated ending up in the similarity score matrix. Subsequently, data is randomly projected onto a lower dimension in the second step. In the third step, the previous similarity is calculated in a lower dimensional space. [MAAT08] The steps can be comprehended based on Figure A-12. While *t-SNE* retains both local and global structure, it is computationally very expensive [KAPR20].

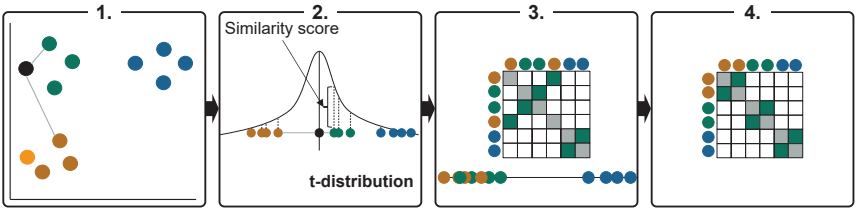


Figure A-12: Concept of t-SNE

Besides *t-SNE*, the following techniques are used for dimensionality reduction: *locally linear embedding* (LLE) [ROWE00], *multidimensional scaling* [KRUS64], also known as *principal coordinates analysis*, *sparse random projection* [ACHL01], *truncated singular value decomposition* [HALK11], *ISOMAP* [TENE00] and *UMAP* [MCIN18] over *self-organizing maps* (SOM) [KOHO82] to *non-matrix factorization* [PAAT94]. [GÉRO18, pp. 223-224]

ML-based

ML based methods for reducing dimensionality are *autoencoders* [RUME86] and *k means clustering* [MACQ67]. *Autoencoders* reconstruct their inputs to outputs and are composed of two parts. The first part consists of an encoder, which narrows down the input into a latent-space representation. Secondly, the decoder again reconstructs the input. After encoding and before decoding exists a bottleneck, which is a representation of the input data set onto a low dimensional space. [GÉRO18, pp. 411-413]

II. Feature selection

Feature selection comprises the choice of features based on filter, or wrapper methods, or alternatively by embedded properties of ML algorithms. When applying filter methods, attributes are removed from the data set based on a certain threshold value. Wrapper techniques than consider not only the data set, but a baseline ML algorithm. Based on how the features influence the performance of the ML model, features are kept or disregarded. Embedded feature selection can be achieved through regularization techniques. Figure A-13 lists 26 different DPP methods.

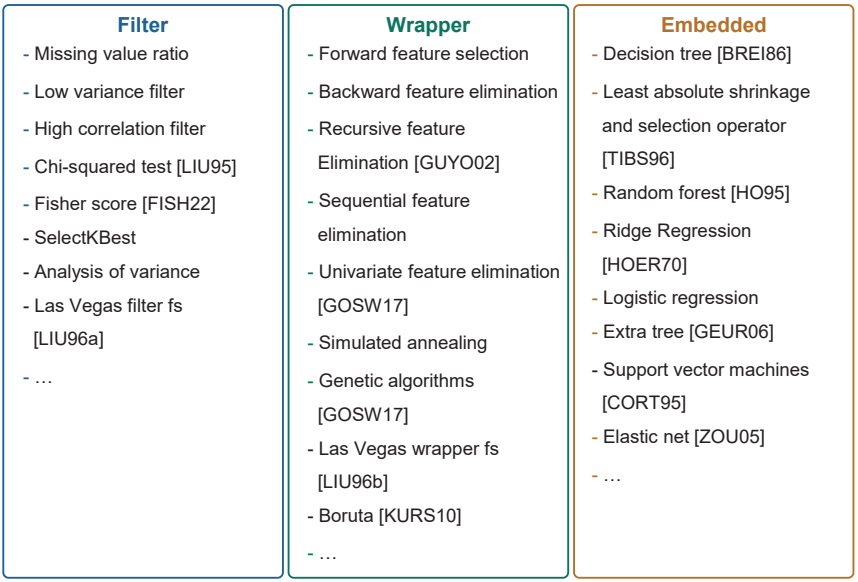


Figure A-13: DPP methods for the DPP category "feature selection"

Filter

Exemplary techniques are *missing value ratio*, *low variance filter* (*VarianceThreshold*) or *high correlation filter* (*RemoveCorrelated*) [FANC21, pp. 5-6]. *Missing value ratio* was already introduced during missing value handling. If the proportion of missing values exceeds a predefined threshold value, columns are removed. *VarianceThreshold* disregard attributes that exhibit only little variance, i. e., which contain data with many constant values and low cardinality. [WIDM15] As all filters, a threshold needs to be set based on which a column is disregarded. The higher the threshold value, the more aggressive is the reduction. WIDMANN ET AL. suggest a threshold of 0.03. WU ET AL.

apply a value of 0.8 [WIDM15], [WUWE19, p. 4]. For successful and meaningful application, the *VarianceThreshold* is to be used on previously scaled data [WIDM15]. Both *high correlation* and *low variance filter* only work with numerical features [BROW19c].

Features with similar values carry similar information. A measure for identifying similarity in the data is correlation. Columns with high correlations do not have a high impact on ML models' performance. [WIDM15] The *high correlation filter* removes highly correlated features based on a threshold value using the correlation coefficient. The coefficient lies between 0 and 1, while 1 indicates a very strong correlation. LIU ET AL. and CHAN suggest to disregard columns in case a correlation coefficient is equal or higher than 0.8 [LIUY20, p. 1775], [CHAN03, p. 614]. Further examples of filters are *chi-squared test* [LIU95], *Fisher score* [FISH22], or *Las Vegas filter* [LIU96a].

Wrapper

While filter approaches solely concentrate on the data set, wrapper methods include baseline ML algorithms to determine feature's impact on ML model performance. Features that contribute least to ML models' predictions are eliminated. [GARC15, pp. 174-175] *Forward feature selection* (*FFS*), *backward feature elimination* (*BFE*) and *recursive feature elimination* (*RFE*) represent prominent examples [FANC21, pp. 5-6], [GUYO03]. *BFE* follows the concept of removing those features, which contribute least to the performance of ML models [GARC15, p. 165], [GUYO03, p. 1167]. First, a model is trained using all features in the data set. Then, one feature is removed at a time and the ML model trained on all remaining features. [FERR94, p. 405] ML models that are usually applied are *k-nearest neighbor* and *random forest* algorithms. The extent of reduction by *BFE* is commonly set to 50 % representing the key hyperparameter of *BFE* [PEDR21a]. In contrast to *BFE*, *FFS* starts with an empty data set and iteratively adds features that result in highest ML model performance increase [GARC15, p. 165].

Within the application of *RFE,* ML models are recursively generated by taking into account continuously decreasing sets of attributes. First, the ML model is trained on an entire set of features and the importance of each feature is obtained. Then, least important features are removed from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features is reached. [GUYO02, p. 394]. In contrast to *BFE* and *FFE, RFE* is computationally less expensive, since it is using sub sets of features, however, it requires ML algorithms that provide the feature importance function [PEDR21f]. Further examples of wrapper methods are *sequential, univariate feature elimination* [GOSW17], *simulated annealing*, *genetic algorithms* [GOSW17], *Las Vegas wrapper* [LIU96b] and *Boruta* [KURS10].

Embedded

Besides filter and wrapper methods, feature selection can be embedded in the ML algorithm. *Decision trees* apply pruning to reduce the size [BREI84]. One or more splits are removed if the sections do not contribute to the model performance. Besides *LASSO* [TIBS96], *ridge regression* also uses regularization by introducing penalty terms [HOER70]. Similarly, *elastic net* introduces a regularization term and performs feature selection by applying the term [ZOU05]. Moreover, feature importance of every tree-based ML algorithm can lead to manually selecting features. The higher the score, the more the feature contributes to the ML model's output. Feature importance is relative, so if one feature is removed from the data set, the feature importance changes again. For both approaches, dimensionality reduction and feature selection, the associated loss of variance has to be taken into account. [GÉRO18, pp. 190-191, 205-223] Further examples are *random forest* [BREI01] or *extra trees* [GEUR06]. In general, *SVMs* [CORT95] are further applied to perform embedded feature selection.

III. Instance selection

Figure A-14 shows a selection of 16 DPP methods for instance selection (filter and wrapper methods) including primary sources, whose could be found. For further reading, the author refers to [GARC15, pp. 195-244].

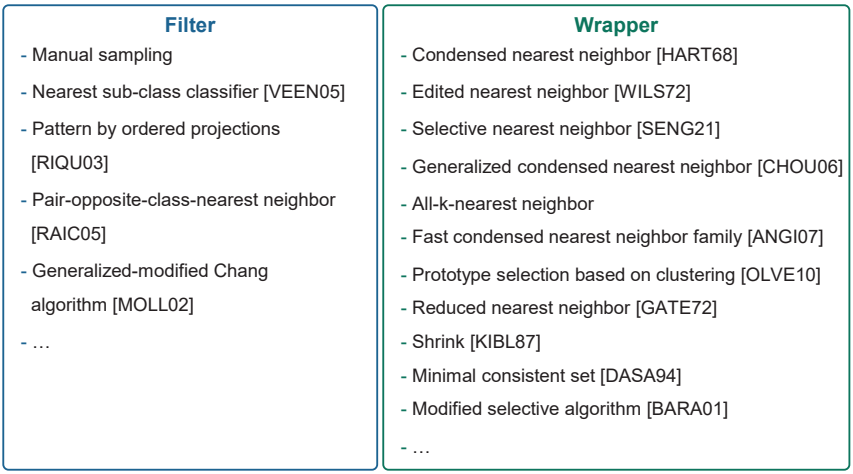


Figure A-14: DPP methods for the DPP category "instance selection" according to [GARC15, pp. 201-206]

Besides manual sampling, *nearest sub-class classifier* [VEEN05], *patterns by ordered projections* [RIQU03], *pair-opposite-class-nearest neighbor* [RAIC05] and *generalized-modified chang algorithm* [MOLL02] are examples of filter approaches for numerosity

reduction. In addition to filter, wrapper approaches can be used for selecting instances. Examples are *condensed nearest neighbor* [HILB67], *edited nearest neighbor* [WILS72], *selective nearest neighbor* [RITT75], *generalized condensed nearest neighbor* [CHOU06], *all-k-nearest neighbor*, *fast condensed nearest neighbor family* [ANGI07], *prototype selection based on clustering* [OLVE10], *reduced nearest neighbor* [GATE72], *shrink* [KIBL87], *minimal consistent set* [DASA94] and *modified selective algorithm* [BARA01].

A.4 Categories and Methods of Data Augmentation and Balancing

By augmenting data, the amount of instances is increased. Balancing data additionally manages the balance between classes. A list of DPP methods is provided for augmentation and balancing in the following.

I. Data augmentation

Data can be augmented for numerical data by applying either *random noise* or *autoencoders* [BISH06, p. 347]. The noise is generated by adding predefined mean and standard deviation values to the original data set. The NumPy library sets the default value of mean to 0 and deviation to 0.1 [NUMP21]. The extent of how much noise should be additionally added to the original data set comprises another hyperparameter. Common values are 20 % meaning that the data set is augmented by up to 20 % [DIET00, p. 10]. Whereas too little noise has nearly no effect on the performance of ML models, too much noise makes it hard to learn the underlying function, thus leading to overfitting [GARC15, p. 22].

Autoencoders cannot only be applied for feature selection, but also for data augmentation. Based on the learning principle of *autoencoders*, inputs are reconstructed. The output of the decoder aims at showing the input as precise as possible, however, will be different to some extent. [GÉRO18, p. 411] These outputs can be added to the original data set. The impact on the ML model performance is dependent on the performance of the *autoencoder*. Well performing *autoencoders* reconstruct their inputs very good, leading to less noise being introduced in the data set.

II. Data balancing

Balancing data is distinguished into over-, under-, as well as hybrid sampling. In Figure A-15, 23 DPP methods are assigned to corresponding sampling section.

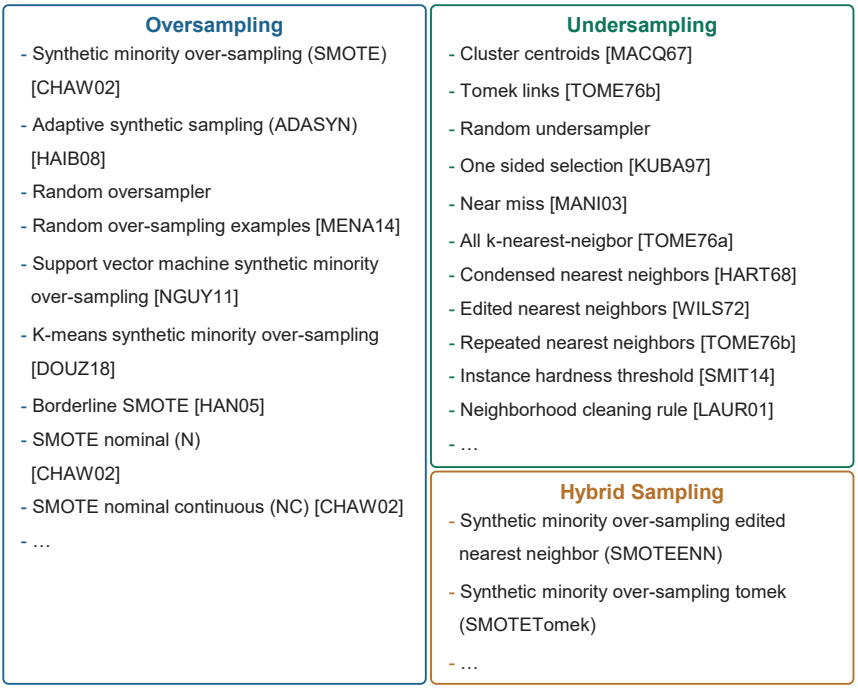


Figure A-15: DPP methods for the DPP category "data balancing" for classification

Oversampling

In case of oversampling, *synthetic minority over-sampling technique* (*SMOTE*) generates new samples by randomly interpolating created data points of the minority class [CHAW02]. For that, a data point is added between a sample of the minority class and its nearest neighbor. The interpolation is performed by calculating the distance between the sample of the minority class and its nearest neighbor, randomly multiplied by a value that lies between 0 and 1. Consequently, a new data point is created that lies between the two data points of the minority class. [CHAW02, p. 328] Noisy data points can be generated since the underlying distribution of data samples is not considered and randomly selected data points can be in a dense distribution [LUEN11, p. 1911]. Many different extensions were developed based on *SMOTE*: *K-Means-SMOTE* [DOUZ18], *SVM-SMOTE* [NGUY11], and *borderline-SMOTE* [HAN05]. In production data sets, a mix of categorical features in its encoded form and numerical columns are available. For these cases, *SMOTENC* was developed, which handles categorical features different from numerical features making it tailored for production use cases [CHAW02]

Adaptive Synthetic Sampling (ADASYN) also generates new samples by interpolation, however, concentrates on generating new data points depending on the local distribution of the underrepresented class [HAIB08]. Another oversampling technique is *RandomOversampler* that creates new samples by randomly sampling with replacement of currently available samples. Through this bootstrapping, resampling of classes is performed by considering each targeted class independently. [MENA14] *RandomOversamplers* are fast, allow for numerical and categorical data. However, data points of the minority class are randomly duplicated. [BATI04, p. 23]

Undersampling

If enough data points exist, data can be undersampled. *Cluster centroids* represents a generation algorithm that reduces the number of samples in the targeted classes but generates the remaining samples. The number of samples are reduced by k means clustering. Each class is synthesized with the centroids of the k means method instead of the original samples. [MACQ67] *Tomek Links* represents a selection algorithm that selects data points from the data set at hand. Since the number of samples is not specified to the user, Tomek Links are assigned to cleaning undersamplers [TOME76a]. *Tomek Links* identify samples of different classes, which are close to each other based on *k-nearest neighbor*. In case nearest neighbors are found, the sample of the majority class is removed. Further derivatives of this approach are *repeated nearest neighbor* [TOME76a] and *all-k-nearest neighbor* [TOME76b], which is further a modification of the *edited nearest neighbor* [WILS72]. *Random undersampler* represents another selection algorithm. The number of samples are specified by the user, therefore, random undersampler is referred to controlling undersampling. Further techniques are *condensed nearest neighbors*, *edited nearest neighbors*, *repeated nearest neighbors* [HART68], *instance hardness threshold* [SMIT14], *near miss* [MANI03], *neighborhood cleaning rule* [LAUR01] and *one sided selection* [KUBA97].

Hybrid sampling

Hybrid sampling combines oversampling and undersampling techniques. For that reason, oversampling is applied prior to undersampling [BATI04, p. 23]. In case of oversampling, *SMOTE* is chosen. For undersampling both *edited nearest neighbor* (ENN) and *Tomek Links* are selected. When combining both techniques individually, SMOTEENN and SMOTETomek are the resulting hybrid sampling techniques. The goal of a subsequent undersampling lies in disregarding those data points, which neighbors are too much oversampled in a class [ZHAX19, p. 458], [BATI04, p. 23]. For *SMOTEEN*, *k-nearest neighbors* are applied to remove wrongly classified data points. In case of *SMOTETomek*, distance measures are used. As introduced, regression

learning tasks also follow balancing techniques, in which *SMOGN* represents a promising yet popular method [BRAN17].

A.5 AutoML systems

Open source: Auto-WEKA by THORNTON ET AL. (2012)

Auto-WEKA, which is based on the *WEKA library* [HALL09, pp. 10-12], addresses the CASH problem by applying *Bayesian optimization* in the form of *sequential model-based optimization* (SMBO) that handles both categorical and continuous hyperparameters. Within *SMBO*, the acquisition function is used to determine the next configuration to gain most knowledge from. In *Auto-WEKA*, the common acquisition function of positive *expected improvement* (EI). The surrogate model can either be a *sequential model based algorithm configuration* (SMAC) with a *)*or a *tree structured parzen estimator* (TPE). [THOR13] Data preprocessing methods are not included into first *Auto-WEKA* versions, however, feature selection methods have been added in later versions. [KOTT13, p. 15]

Open source: H2O AutoML by LEDELL & POIRIER (2020)

H2O creates ML pipelines from a set of DPP techniques and ML algorithms. Hyperparameter optimization within the CASH problem is carried out by either random or grid search. *H2O* supports many ML algorithms, namely: *random forest*, *SVM*, *Naïve Bayes classifier*, *XGBoost*, *generalized linear models* and *additive models*, as well as *deep learning neural networks*. DPP methods are only available for *XGBoost*, which comprise *imputation*, *scaling*, and *one hot encoding*. In case of *H2O*, encoding is not required since categorical features are handled natively by tree based algorithms. Further DPP methods such as feature selection are planned for future releases. [LEDE20], [H2O16].

Further open source AutoML approaches

ML box automatically identifies DPP pipelines by applying *tree parzen estimator* as optimizer. However, DPP methods are limited to basic cleaning, encoding, and feature selection techniques. [ARON20] The limited amount of DPP methods also applies for *Hyperopt-Sklearn* [KOME19], *AutoPytorch Tabular* [ZIMM20], *auto_ml* [PARR21], *AutoGluon* [ERIC20] and *Uber Ludwig* [MOLI19]. *TransmogrifAI* [MOOE21] provides MV imputation, encoding, scaling, feature selection, and balancing. As for other AutoML systems, *AutoKeras* [HAIF19] covers just a few DPP methods. Further AutoML tools can be found in [KRAU20].

A.6 Production Data Sets and Computing Platforms

For the development of Meta-DPP, 44 publicly available data sets and 6 data sets from further projects are used. The data sets including its derivatives of *norm, both, cats,* and *mvs* are listed in Table A-1-Table A-6.

For every data set, the corresponding learning task (*LT*), number of instances (*Inst.*), number of attributes (*Attr.*), number of categorical features (*Cats*), number of missing values (*MV*), the class representation in case of classification and standard deviation of the target variable in case of regression (*CL / STD*) as well as the application area (*App. Area*) and reference (*Ref.*) is depicted.

All three application areas are considered for developing Meta-DPP. In case of *process*, the use cases of process design (*PD*), product quality prediction (*PQ*). For *machines & assets*, predictive maintenance (*PdM*) and anomaly detection (*AD*) are used. In case of *product*, the product design is chosen.

The 44 publicly available data sets are 3D Printer [OKUD19], Condition monitoring hydraulic systems (cooler, valve, pump and accumulator) [HELW15], Airfoil Self-Noise [DUA19], APS Failure at Scania Trucks [DUA19], Bolts [STEP14], [VANS13], [FEUR21], CNC Mill Tool Wear [UMIC18], Maintenance of Naval Propulsion Plants Compressor as well as Maintenance of Naval Propulsion Plants Turbine [CORA14], Cylinder Bands [DUA19], Energy Optimization Anomalous Optimized (EOAO), Energy Optimization Anomalous Standard (EOAS) [INIT18], [BIRG17], [BIRG18c], [HRAN16], Flight Software for Earth Orbiting Satellite (FSEOS) and Flight Software for Earth Orbiting Satellite (1) (FSEOS(1)) [SAYY05], [VANS13], [FEUR21], Genesis Demonstrator Classification Anomaly Detection (GDAD), Genesis Demonstrator Classification Machine State (GDMS), Genesis Demonstrator Multiclass (GDM) [INIT18], [BIRG18b], Laser Welding [RINN20], Mechanical Analysis Data Set [DUA19], Mercedes Benz Greener Manufacturing [MERC20], Milling (Table and Spindle) [AGOG07], Pulsar Star [LYON16], Robot Execution Failures [DUA19], SECOM [DUA19], Software for ground data (SFGD) [SAYY05], [VANS13], [FEUR21], Steel Plates Faults [SEME22], Turbofan Engine Degradation Simulation [SAXE08], Unknown 1 and Unknown 2 [VANS13], [FEUR21]. Subsequently, the meta data of the six data sets from former projects are listed.

Table A-1: Data sets used for developing Meta-DPP (1/6)

No.	Data Set	LT	Inst.	Attr.	Cats	MV	CL / STD	App. Area	Ref.
001	3D Printer	RE	50	11	2	0	0.7802	PD	OKUD19
002	3D Printer Elongation	RE	50	9	2	0	0.7802	PD	OKUD19
003	3D Printer Roughness	RE	50	9	2	0	98.04	PD	OKUD19
004	3D Printer Strength	RE	50	9	2	0	8.836	PD	OKUD19
005	Accumulator	CL	2,205	9	0	0	808:399:399:599	PdM/PQ	HELW15
006	Accumulator both	CL	2,205	9	4	794	808:399:399:599	PdM/PQ	HELW15
007	Accumulator cats	CL	2,205	9	4	0	808:399:399:599	PdM/PQ	HELW15
008	Accumulator mvs	CL	2,205	9	0	1,588	808:399:399:599	PdM/PQ	HELW15
009	Airfoil Self-Noise	RE	1,503	5	0	0	15.72	PD	DUA19
010	Airfoil Self-Noise both	RE	1,503	5	4	451	15.72	PD	DUA19
011	Airfoil Self-Noise cats	RE	1,503	5	4	0	15.72	PD	DUA19
012	Airfoil Self-Noise mvs	RE	1,503	5	0	451	15.72	PD	DUA19
013	Airfoil Self-Noise SPL	RE	1,503	5	0	451	6.896	PD	DUA19
014	Airfoil Self-Noise SPL both	RE	1,503	5	4	451	6.896	PD	DUA19
015	Airfoil Self-Noise SPL cats	RE	1,503	5	4	0	6.896	PD	DUA19
016	Airfoil Self-Noise SPL mvs	RE	1,503	5	0	451	6.896	PD	DUA19
017	APS	CL	76,000	170	0	1,078,695	74,625:1,375	PdM/PQ	SCAN16
018	Bolts	CL	40	7	0	0	14:26	PdM/AD	STEP14
019	Bolts both	CL	40	7	4	11	14:26	PdM/AD	STEP14
020	Bolts cats	CL	40	7	4	0	14:26	PdM/AD	STEP14
021	Bolts mvs	CL	40	7	0	11	14:26	PdM/AD	STEP14
022	CNC Mill Tool Wear	CL	25,286	48	1	0	11,978:13,308	PQ	UMIC18

Table A-2: Data sets used for developing Meta-DPP (2/6)

No.	Data Set	LT	Inst.	Attr.	Cats	MV	CL / STD	App. Area	Ref.
023	Compressor	RE	11,934	16	0	0	0.0147	PdM	CORA14
024	Compressor both	RE	11,934	16	5	9,547	0.0147	PdM	CORA14
025	Compressor cats	RE	11,934	16	5	0	0.0147	PdM	CORA14
026	Compressor mvs	RE	11,934	16	0	11,457	0.0147	PdM	CORA14
027	Cooler	CL	2,205	393	0	0	732:732:741	PdM/ PQ	HELW15
028	Cylinder Bands	CL	17,898	8	0	0	16,259 :1,639	PQ	DUA19
029	EOAO	CL	19,634	19	0	0	15,117 :4,517	PQ/ AD	INIT18
030	EOAS	CL	23,645	19	0	0	17,975 :5,670	PQ/ AD	INIT18
031	Flag	CL	2,205	25	0	0	1,449: 756	PdM /PQ	HELW15
032	Flag both	CL	2,205	25	6	2,205	1,449: 756	PdM /PQ	HELW15
033	Flag cats	CL	2,205	25	4	0	1,449: 756	PdM /PQ	HELW15
034	Flag mvs	CL	2,205	25	0	2,756	1,449: 756	PdM /PQ	HELW15
035	FSEOS	CL	1,109	21	0	0	1,032: 77	AD	SAYY05
036	FSEOS both	CL	1,109	21	17	1,397	1,032: 77	AD	SAYY05
037	FSEOS cats	CL	1,109	21	4	0	1,032: 77	AD	SAYY05
038	FSEOS mvs	CL	1,109	21	0	1,397	1,032: 77	AD	SAYY05
039	FSEOS (1)	CL	5,589	36	0	0	5,566: 23	AD	SAYY05
040	FSEOS (1) both	CL	5,589	36	4	10,060	5,566: 23	AD	SAYY05
041	FSEOS (1) cats	CL	5,589	36	6	0	5,566: 23	AD	SAYY05
042	FSEOS (1) mvs	CL	5,589	36	0	12,073	5,566: 23	AD	SAYY05
043	GDAD	CL	16,220	19	0	0	16,170 :39:11	AD	INIT18
044	GDAD both	CL	16,220	19	4	15,409	16,170 :39:11	AD	INIT18

Table A-3: Data sets used for developing Meta-DPP (3/6)

No.	Data Set	LT	Inst.	Attr.	Cats	MV	CL / STD	App. Area	Ref.
045	GDAD cats	CL	16,220	19	4	0	16,170 :39:11	AD	INIT18
046	GDAD mvs	CL	16,200	19	0	15,409	16,170 :39:11	AD	INIT18
047	GDM	CL	22,940	24	0	0	7,424: 7,040: 8,476	AD	INIT18
048	GDMS	CL	16,220	19	0	0	1,882: 55:76: 383:4, 518:94 2:5,19 3:2,22 9:942	AD	INIT18
049	Laser Welding	CL	360	12	0	3	310:50	PD	RINN20
050	Laser Welding Depth	RE	360	12	0	3	165.6	PD	RINN20
051	Mechanical Analysis	CL	9,254	7	1	0	2,931: 3,194: 720:77 3:669: 967	AD	DUA19
052	Mercedes	RE	4,209	377	8	0	12.68	PD	MERC17
053	Milling	RE	167	12	0	21	0.0211	PdQ/ PM	AGOG07
054	Milling Spindle	RE	167	11	0	21	0.0211	PdQ/ PM	AGOG07
055	Milling Table	RE	167	11	0	21	0.0246	PdQ/ PM	AGOG07
056	Pulsar Star	CL	17,898	8	0	0	16,259 :1639	AD	LYON16
057	Pulsar Star both	CL	17,898	8	4	7159	16,259 :1639	AD	LYON16
058	Pulsar Star cats	CL	17,898	8	4	0	16,259 :1639	AD	LYON16
059	Pulsar Star mvs	CL	17,898	8	0	5727	16,259 :1639	AD	LYON16
060	Pump	CL	2,205	5	0	0	1,221: 492: 492	PdM/ PQ	HELW15
061	Pump both	CL	2,205	5	4	992	1,221: 492: 492	PdM/ PQ	HELW15
062	Pump cats	CL	2,205	5	4	0	1,221: 492: 492	PdM/ PQ	HELW15

Table A-4: Data sets used for developing Meta-DPP (4/6)

No.	Data Set	LT	Inst.	Attr.	Cats	MV	CL / STD	App. Area	Ref.
063	Pump mvs	CL	2,205	5	0	882	1,221:492:492	PdM/PQ	HELW15
064	Robot Execution Failures	CL	6,945	6	0	0	105:390:315:1,335:705:390:240:90:135:45:225:1,935:825:75:135	PM	DUA19
065	SECOM	CL	1,567	590	0	41,951	1,463:104	PQ	DUA19
066	SECOM both	CL	1,567	591	7	85,874	1,463:104	PQ	DUA19
067	SECOM cats	CL	1,567	591	6	41,650	1,463:104	PQ	DUA19
068	SECOM mvs	CL	1,567	591	1	86,223	1,463:104	PQ	DUA19
069	SFGD	CL	2,109	21	0	0	1,783:326	AD	SAYY05
070	SFGD both	CL	2,109	21	4	1,772	1,783:326	AD	SAYY05
071	SFGD cats	CL	2,109	21	9	0	1,783:326	AD	SAYY05
072	SFGD mvs	CL	2,109	21	0	2,214	1,783:326	AD	SAYY05
073	Steel Plates Faults	CL	1,941	27	0	0	158:190:391:72:55:402:673	AD	SEME22
074	Steel Plates Faults both	CL	1,941	27	5	3,668	158:190:391:72:55:402:673	AD	SEME22
075	Steel Plates Faults cats	CL	1,941	27	8	0	158:190:391:72:55:402:673	AD	SEME22
076	Steel Plates Faults mvs	CL	1,941	27	0	4,717	158:190:391:72:55:402:673	AD	SEME22

Table A-5: Data sets used for developing Meta-DPP (5/6)

No.	Data Set	LT	Inst.	Attr.	Cats	MV	CL / STD	App. Area	Ref.
077	Turbine	RE	11,934	16	0	0	0.0075	PdM	CORA14
078	Turbine both	RE	11,934	16	6	7,638	0.0075	PdM	CORA14
079	Turbine cats	RE	11,934	16	5	0	0.0075	PdM	CORA14
080	Turbine mvs	RE	11,934	16	0	11,457	0.0075	PdM	CORA14
081	Turbofan 1	RE	13,096	26	0	0	41.39	PdM/ AD	SAXE08
082	Turbofan 2	RE	33,991	26	0	0	49.17	PdM/ AD	SAXE08
083	Turbofan 3	RE	16,596	26	0	0	41	PdM/ AD	SAXE08
084	Turbofan 4	RE	41,214	26	0	0	51.2	PdM/ AD	SAXE08
085	Unknown 1	CL	9,466	38	0	0	9,398: 68	AD	VANS13
086	Unkown 1 both	CL	9,466	38	5	17,986	9,398: 68	AD	VANS13
087	Unknown 1 cats	CL	9,466	38	6	0	9398:6 8	AD	VANS13
088	Unknown 1 mvs	CL	9,466	38	0	14,388	9,398: 68	AD	VANS13
089	Unknown 1 Target	CL	9,466	38	0	0	9,398: 68	PQ	VANS13
090	Unknown 2	CL	161	39	0	0	109:52	AD	VANS13
091	Unknown 2 both	CL	161	39	7	565	109:52	AD	VANS13
092	Unknown 2 cats	CL	161	39	7	0	109:52	AD	VANS13
093	Unknown 2 mvs	CL	161	39	0	314	109:52	AD	VANS13
094	Unknown 2 Target	CL	161	39	0	0	109:52	PQ	VANS13
095	Valve	CL	2,205	38	0	0	360:36 0:360: 1,125	PdM/ PQ	HELW15
096	Valve both	CL	2,205	38	6	5,027	360:36 0:360: 1,125	PdM/ PQ	HELW15
097	Valve cats	CL	2,205	38	5	0	360:36 0:360: 1,125	PdM/ PQ	HELW15
098	Valve mvs	CL	2,205	38	0	2,253	360:36 0:360: 1,125	PdM/ PQ	HELW15

Table A-6: Data sets used for developing Meta-DPP (6/6)

No.	Data Set	LT	Inst.	Attr.	Cats	MV	CL / STD	App. Area	Ref.
099	Project data 1	CL	642	167	6	1,041	29:613	PQ	IPT
100	Project data 2	CL	1,111	39	20	16	1,108:2	PQ	IPT
101	Project data 3	CL	1,484	200	4	7,661	1,427:57	PQ	IPT
102	Project data 4	CL	784	79	3	105	781:3	PQ	IPT
103	Project data 5	CL	683	381	3	2,603	675:8	PQ	IPT
104	Project data 6	CL	964	357	10	522	918:46	PQ	IPT

Table A-7 shows nine computing platforms used for benchmarking DPP pipelines including the operating system (OS), random-access memory (RAM), CPU and number of cores. Two platforms are stand-alone machines located at the Fraunhofer IPT. Four virtual machines are used, which are located at the virtual fort knox (VFK) at the Fraunhofer IPT. In addition, simulations were performed with computing resources granted by RWTH Aachen University under project ID 4474 as well as ID 4941. The last computing platform is a virtual machine located at Hochschule Furtwangen.

Table A-7: Overview of computing platforms being used for benchmarking

PF	Host	OS	RAM [GB]	CPU	No. of cores
01	Stand-Alone IPT	Windows	384	Intel® Xeon® Gold 5218 CPU @ 2.30 GHz	64
02	Stand-Alone IPT	Windows	384	Intel® Xeon® Silver 4216 CPU @ 2.10 GHz	64
03	VFK	Linux	32	Intel® Xeon® CPU E5-2680 v4 @ 2.40 GHz	16
04	VFK	Linux	16	Intel® Xeon® CPU E5-2680 v4 @ 2.40 GHz	8
05	VFK	Linux	8	Intel® Xeon® CPU E5-2680 v4 @ 2.40 GHz	2
06	RWTH	Linux	64	Intel® Xeon® Platinum 8160 CPU @ 2.10 GHz	32
07	RWTH	Linux	64	Intel® Xeon® Platinum 8160 CPU @ 2.10 GHz	16
08	RWTH	Linux	394	Intel® Xeon® Platinum 8160 CPU @ 2.10 GHz	48
09	HFT	Windows	64	Intel® Xeon® Gold 6226R CPU @ 2.90 GHz	8

PF Platform **HFT** Hochschule Furtwangen **VFK** Virtual Fort Knox

In addition, a benchmarking at small scale is conducted to preselect DPP methods. Therefore, ten data set are used, which can be seen in Table A-8.

Table A-8: Data sets for benchmarking at small scale

No.	Data Set	LT	Inst.	Attr.	Cats	MV	CL / STD	Ref.
01	Milling table	RE	167	11	0	21	0.02463	AGOG07
02	Airfoil Self-Noise	RE	1,503	5	0	0	15.72	DUA19
03	Compressor	RE	11,934	16	0	0	0.01472	CORA14
04	FSEOS	CL	1,109	21	0	0	1032:77	SAYY05
05	FSEOS(1)	CL	5,589	36	0	0	5566:23	SAYY05
06	Unknown 1	CL	9,466	38	0	0	9398:68	SAYY05
07	Unknown 2	CL	161	39	0	0	109:52	SAYY05
08	SFGD	CL	2,109	21	0	0	1783:326	SAYY05
09	Steel Plates Faults	CL	1,941	27	0	0	158:190:391:72:55:402:673	SEME98
10	Pulsar Star	CL	17,898	8	0	0	16259:1639	LYON16

LT	Learning Task	Ref.	Reference	CL / STD	CL (Classification): Ratio of classes
Inst.	No. of instances	MV	No. of missing values		STD: standard deviation of label (Regression)
Attr.	No. of attributes	Cats	No. of categorical cols		

A.7 Determination of DPP methods for Data Cleaning

In the following, all assessments regarding the determination of DPP methods for data cleaning are presented. For every of the seven identified selection criteria, the assessment scores between 0, 0.25, 0.50, 0.75, 1.0 are depicted as Harvey balls. Given the highest scores, the selected DPP methods are shown. Subsequently to the assessments, the hyperparameters of DPP methods are discussed. First, the determination of suitable methods for missing value handling is presented followed by the outlier detection and noisy data handling.

(I) Data cleaning – noisy data handling

Table A-9 and Table A-10 show the assessments for missing value handling.

Table A-9: Determination of suitable methods for missing value handling (1/2)

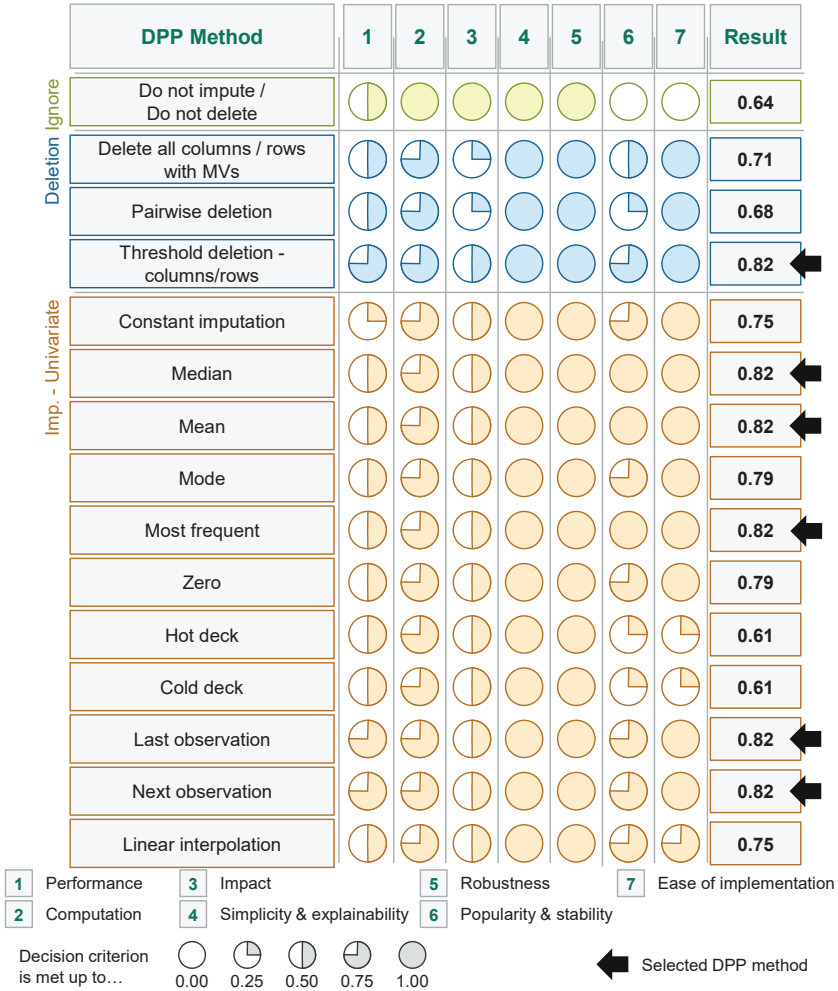
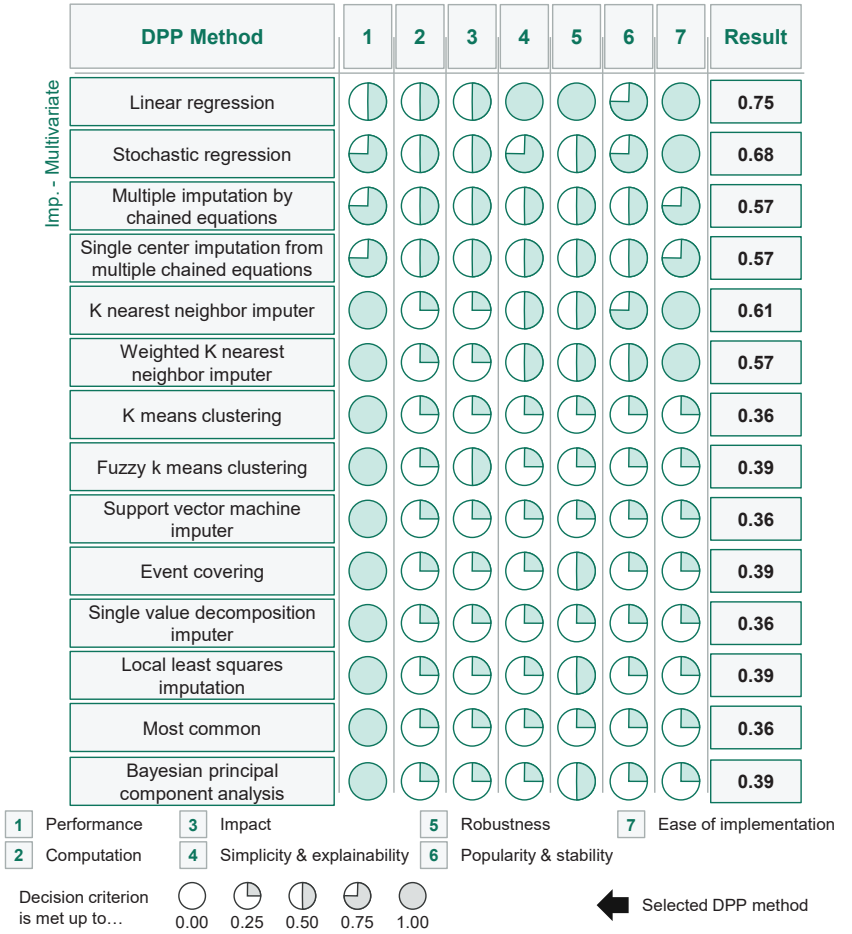


Table A-10: Determination of suitable methods for missing value handling (2/2)



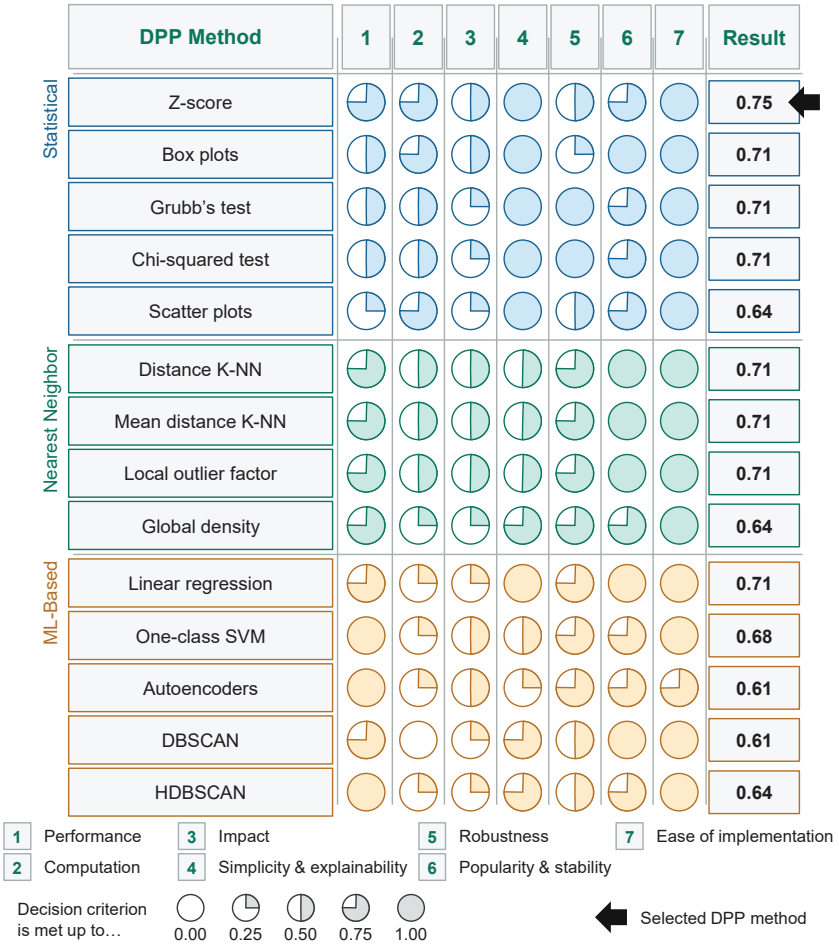
Hyperparameter settings

MV ratio is selected as DPP method. The threshold based on which MVs are removed from the data set, is set to 100 %. Only if the whole column contains missing values, the column is deleted to ensure that no data reduction is realized by applying this method. All other methods are implement with their default hyperparameters.

(II) Data cleaning – outlier detection and handling

Table A-11 shows the assessment scores for outlier detection methods. *Z score* is chosen and implemented with the *MeanImputer*.

Table A-11: Determination of suitable DPP methods for outlier detection



Hyperparameter settings

If the *z score* is higher than 3.0, the data point is marked as outlier. Therefore, the hyperparameter is set to 3. The *MeanImputer* is implemented in its default hyperparameters.

(III) Data cleaning – noisy data handling

Noisy data handling techniques are selected given the assessment scores from Table **A-12**. Every selected DPP method is implemented in its default hyperparameter settings.

Table A-12: Determination of suitable DPP methods for noisy data handling

A.8 Determination of DPP methods for Data Transformation

The assessment scores of data encoding, feature scaling, and handling skewness are shown in the following. Since data discretization is not focused in this dissertation, the assessment is not performed for this DPP category.

(I) Data transformation – data encoding

Table A-13 and Table A-14 shows the assessments for the encoding techniques. Every selected encoding method is implemented in its default hyperparameters.

Table A-13: Determination of suitable DPP methods for data encoding (1/2)

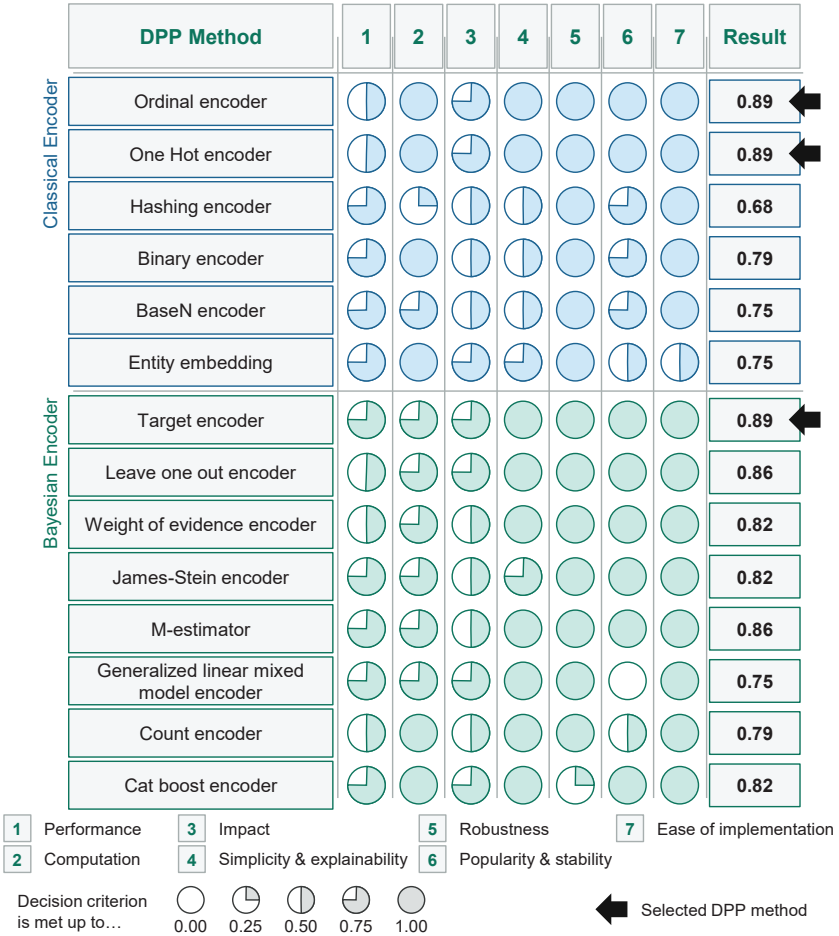
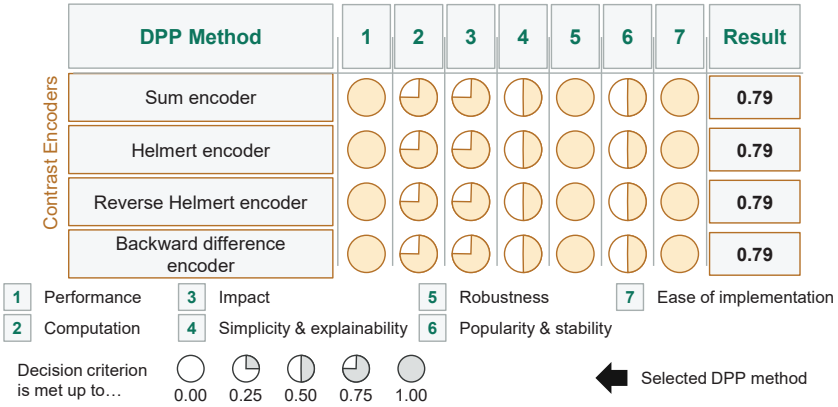


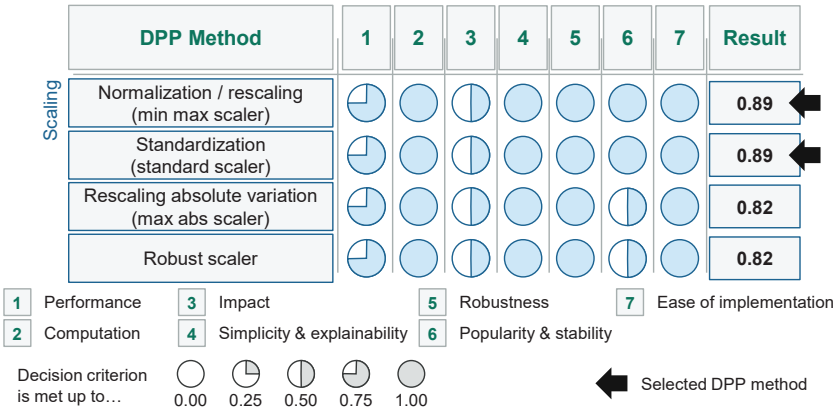
Table A-14: Determination of suitable DPP methods for data encoding (2/2)



(II) Data transformation - feature scaling

For feature scaling, the *MinMaxScaler* and *StandardScaler* are selected in its default hyperparameters (see Table A-15).

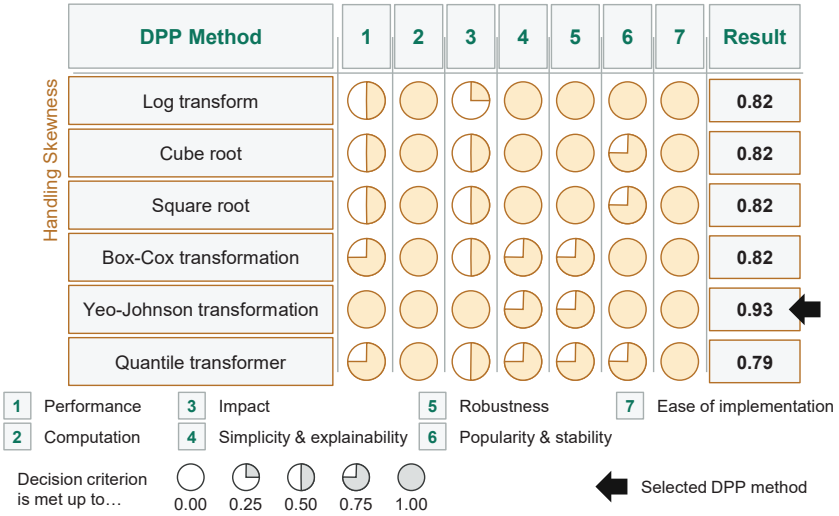
Table A-15: Determination of suitable DPP methods for feature scaling



(III) Data transformation – handling skewness

In case the skewness is handled, the *Yeo-Johnson PowerTransformer* is chosen (see Table A-16).

Table A-16: Determination of suitable DPP methods for handling skewness



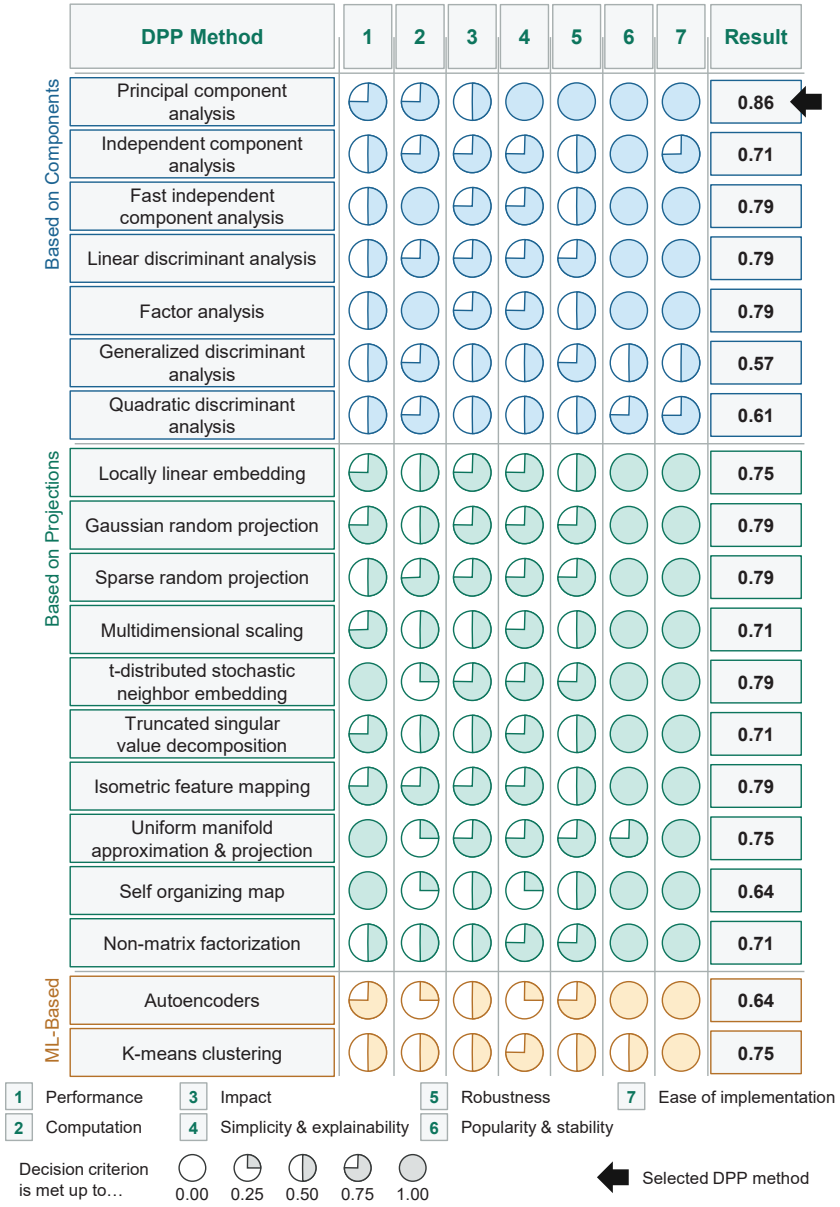
A.9 Determination of DPP methods for Data Reduction

In the following, data reduction methods are selected based on the assessment scores and the set hyperparameters discussed starting with dimensionality reduction, followed by feature selection. The embedded methods are not included in this assessment, since they are not focused within this work.

(I) Data reduction – dimensionality reduction

The assessment scores for dimensionality reduction methods are depicted in Table A-17. When applying the criteria, *PCA* is selected. Critical hyperparameter is the percentage of variance, which is retained by the principal components. The percentage of variance is set to 95 %, i. e., 95 % of the variance of the data is retained by *PCA*.

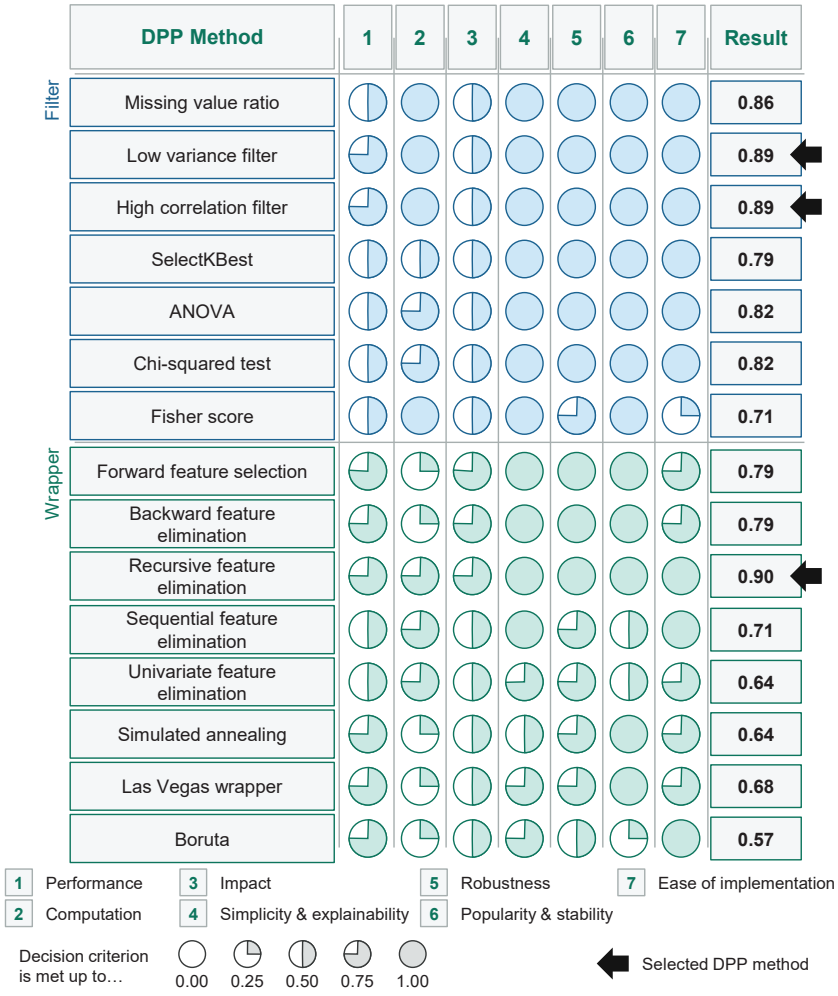
Table A-17: Determination of suitable DPP methods for dimensionality reduction



(II) Data reduction – feature selection

The assessment scores for feature selection can be found in Table A-18. The hyperparameter of the threshold of the correlation coefficient based on which correlated features are removed, will be set to 0.8. Besides high correlation, filtering is also applied by low variance of the features.

Table A-18: Determination of suitable DPP methods for feature selection



A value for low *variance filter*, called *VarianceThreshold* is 0.6, following the same consideration as for *correlation filter*. Therefore, *RFE* is included for configuring DPP pipelines. In case of classification tasks, the baseline ML algorithm being trained during *RFE* is *RF classifier*. When a regression task is present, *RF regressor* is called. In both cases, default hyperparameters are used.

A.10 Determination of DPP methods for Data Augmentation & Balancing

No assessment is necessary for data augmentation, since *RandomNoise* has already been selected. For *random noise*, the mean is specified to $\mu = 0$ and standard deviation to $\sigma = 0.1$ to introduce minimum noise into the data set. Lastly the amount of data points to be added is set as up to 20 %. Table A-19 and Table A-20 show the assessment scores for data balancing.

Table A-19: Determination of suitable DPP methods for data balancing (1/2)

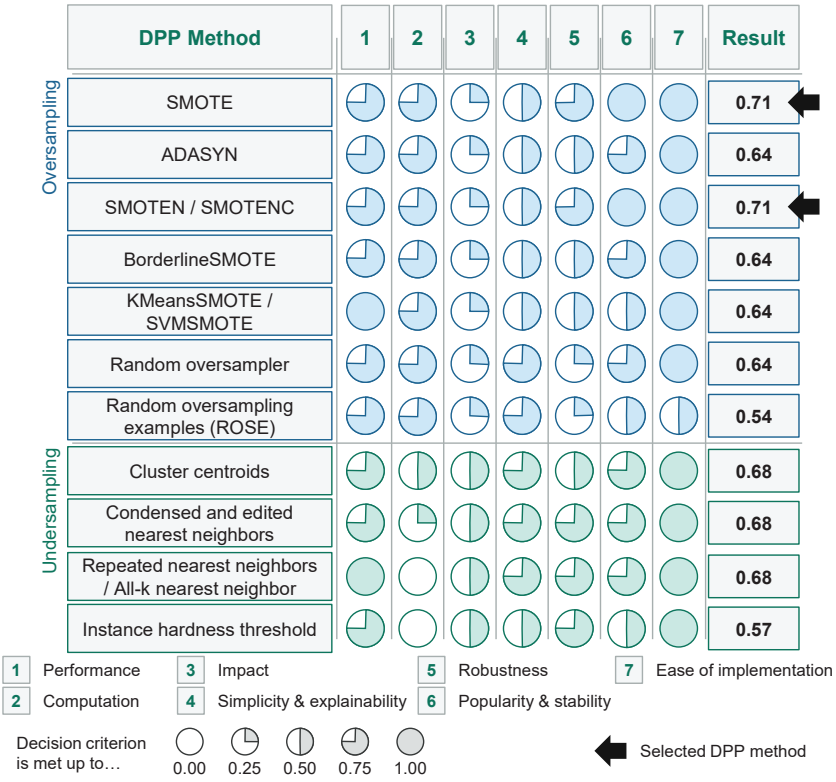
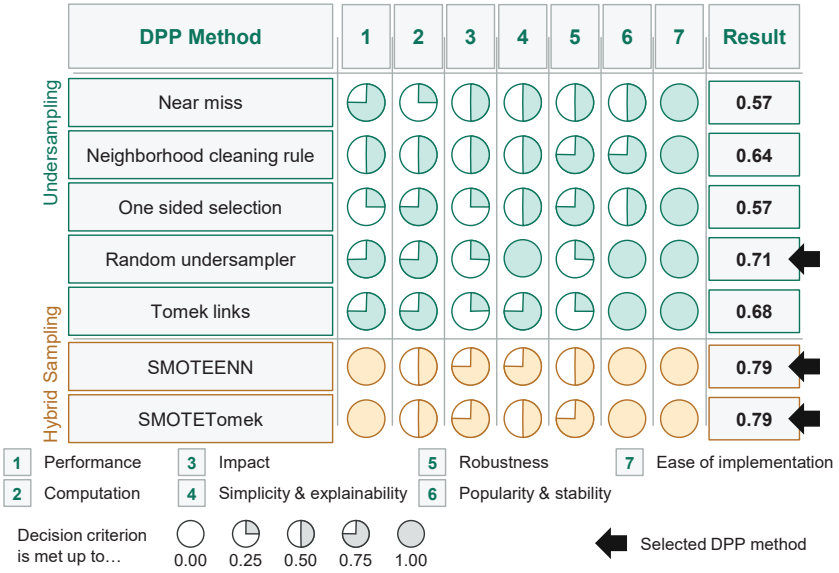


Table A-20: Determination of suitable DPP methods for data balancing (2/2)



In case of data balancing, *SMOTE*, *SMOTENC* (in case of categorical features), random undersampler are used. For hybrid sampling, a benchmarking at small scale is conducted (see Table A-21). Ultimately, *SMOTEENN* is selected.

Table A-21: Results of individual benchmarking for hybrid sampling techniques

	CL - F1 Score		
DPP Method	RF	SVM	MLP
SMOTEENN	0.951	0.735	0.684
SMOTETomek	0.939	0.672	0.651

A.11 Benchmarking and Pooling Results

In the following the benchmarking results are depicted. While Figure A-16 shows data set-specific F1 scores for classification, Figure A-17 provides the overview of NRMSE values over all data sets for regression. Consecutively, Figure A-18 shows ML algorithm specific F1 scores for classification data set. Figure A-19 depicts the NRMSE values for the 18 ML algorithms used during benchmarking. Individual marker show the performance of individual pipelines (both essential and pipeline pool). Figure A-20 and Figure A-21 then show the box plots exemplarily for regression and regression use cases.

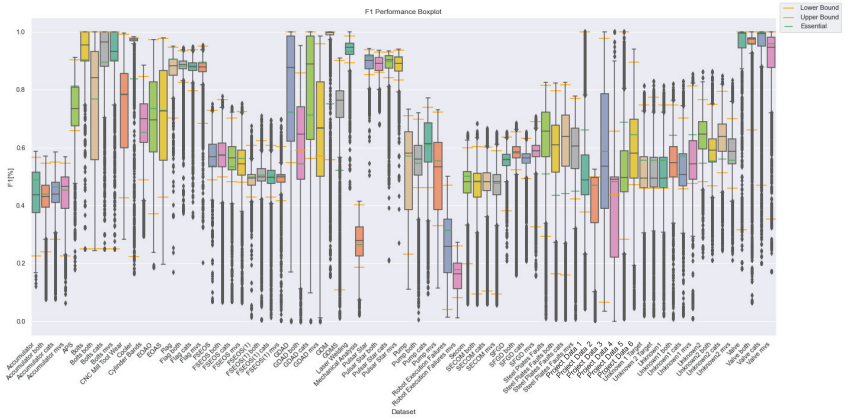


Figure A-16: Data set-specific F1 scores for classification

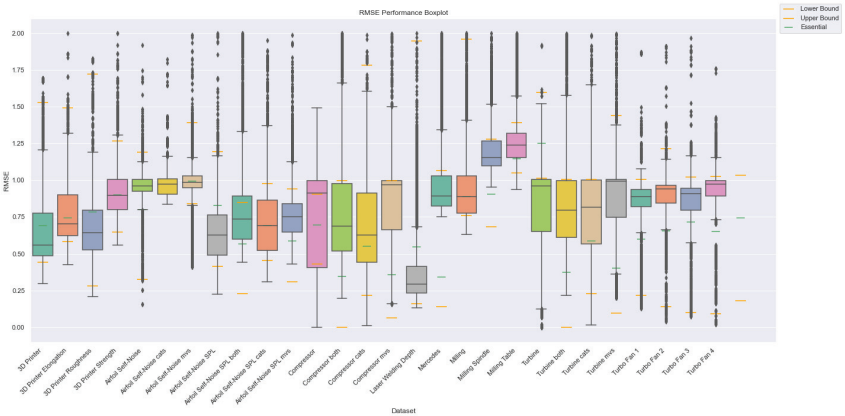


Figure A-17: Data set-specific NRMSE performances for regression

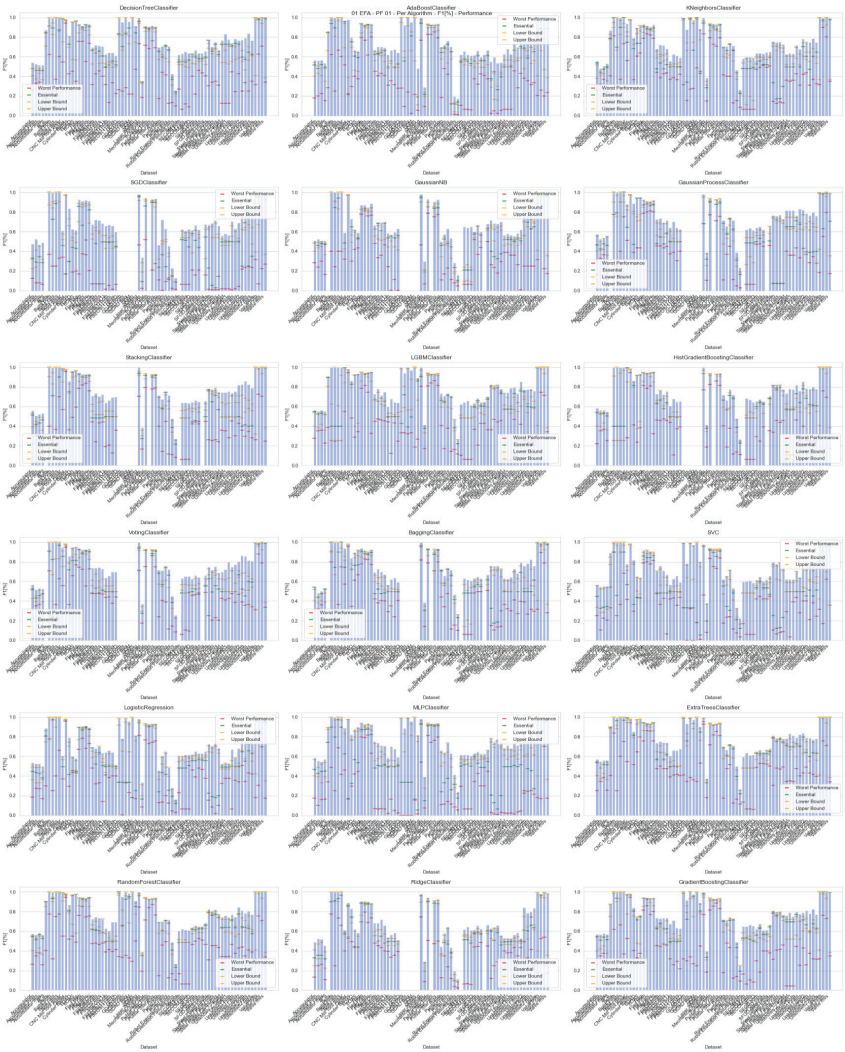


Figure A-18: ML algorithm-specific F1 scores for classification data sets (excerpt)

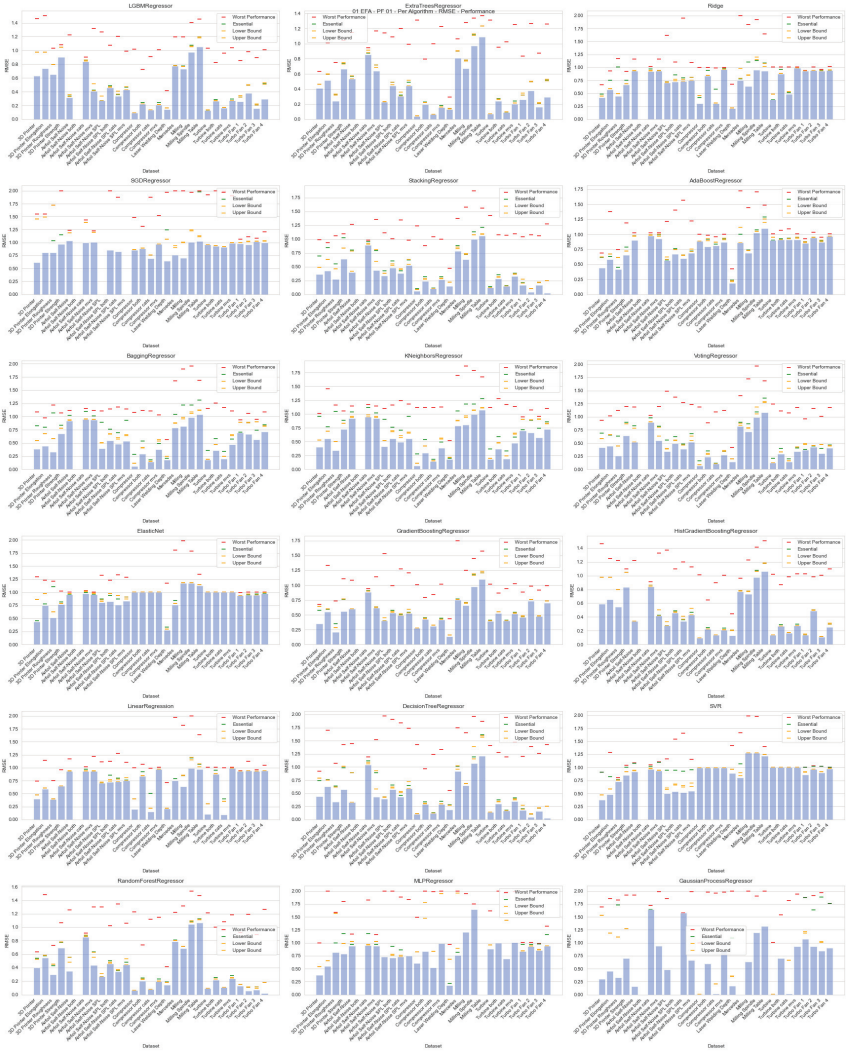


Figure A-19: ML algorithm-specific NRMSE values for regression (excerpt)

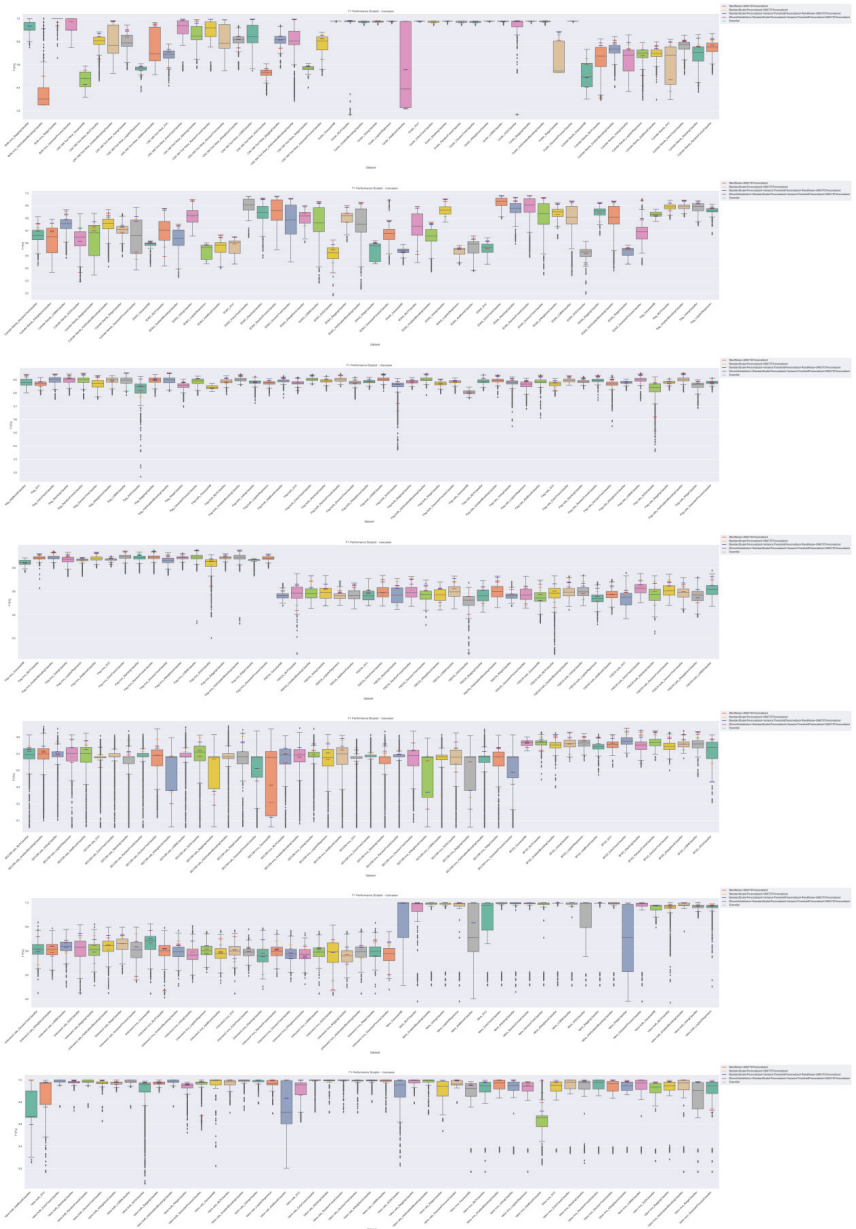


Figure A-20: Exemplary overview of 400 out of 1,278 classification use cases

Figure A-21: Exemplary overview of 350 of 514 regression use cases

A.12 Meta Features Database

The meta features database contains data set-, ML algorithm-, and DPP pipeline-related meta features that is illustrated in Table A-22 - Table A-25. The tables show the dimension (*Dim*), e. g., data set (DS), group (*Gr*), e. g., landmarking (LM), and learning task (*LT*), whether it is assigned to classification (CL) or regression (RE).

Table A-22: Meta features database (1/4)

ID	Name	Dim	Gr	LT	ID	Name	Dim	Gr	LT
001	attr_to_inst	DS	Gen	CL+RE	025	eigenvalues.sd	DS	Stat	CL+RE
002	cat_to_num	DS	Gen	CL+RE	026	g_mean.mean	DS	Stat	CL+RE
003	freq_class.mean	DS	Gen	CL+RE	027	g_mean.sd	DS	Stat	CL+RE
004	freq_class.sd	DS	Gen	CL+RE	028	gravity	DS	Stat	CL+RE
005	inst_to_attr	DS	Gen	CL+RE	029	h_mean.mean	DS	Stat	CL+RE
006	n_continuous_features	DS	Gen	CL+RE	030	h_mean.sd	DS	Stat	CL+RE
007	n_int_features	DS	Gen	CL+RE	031	iq_range.mean	DS	Stat	CL+RE
008	nr_attr	DS	Gen	CL+RE	032	iq_range.sd	DS	Stat	CL+RE
009	nr_bin	DS	Gen	CL+RE	033	kurtosis.mean	DS	Stat	CL+RE
010	nr_cat	DS	Gen	CL+RE	034	kurtosis.sd	DS	Stat	CL+RE
011	nr_inst	DS	Gen	CL+RE	035	lh_trace	DS	Stat	CL+RE
012	nr_num	DS	Gen	CL+RE	036	mad.mean	DS	Stat	CL+RE
013	num_mvs	DS	Gen	CL+RE	037	mad.sd	DS	Stat	CL+RE
014	num_outliers_target	DS	Gen	RE	038	max.mean	DS	Stat	CL+RE
015	kurt_targ	DS	Gen	RE	039	max.sd	DS	Stat	CL+RE
016	rep_classes	DS	Gen	CL	040	mean.mean	DS	Stat	CL+RE
017	nr_class	DS	Gen	CL	041	mean.sd	DS	Stat	CL+RE
018	can_cor.mean	DS	Stat	CL+RE	042	median.mean	DS	Stat	CL+RE
019	can_cor.sd	DS	Stat	CL+RE	043	median.sd	DS	Stat	CL+RE
020	cor.mean	DS	Stat	CL+RE	044	min.mean	DS	Stat	CL+RE
021	cor.sd	DS	Stat	CL+RE	045	min.sd	DS	Stat	CL+RE
022	cov.mean	DS	Stat	CL+RE	046	nr_cor_attr	DS	Stat	CL+RE
023	cov.sd	DS	Stat	CL+RE	047	nr_disc	DS	Stat	CL+RE
024	eigenvalues.mean	DS	Stat	CL+RE	048	nr_norm	DS	Stat	CL+RE

Dim	Dimension	Gr	Group	LT	Learning task	DS	Data set	Gen	General	Stat	Statistical
CL	Classification	RE	Regression								

Table A-23: Meta features database (2/4)

ID	Name	Dim	Gr	LT
049	nr_outliers	DS	Stat	CL+RE
050	num_outliers_boxplot	DS	Stat	CL+RE
051	num_outliers_Zscore	DS	Stat	CL+RE
052	p_trace	DS	Stat	CL+RE
053	range.mean	DS	Stat	CL+RE
054	range.sd	DS	Stat	CL+RE
055	roy_root	DS	Stat	CL+RE
056	sd_mean	DS	Stat	CL+RE
057	sd.sd	DS	Stat	CL+RE
058	sd_ratio	DS	Stat	CL+RE
059	skew_targ	DS	Stat	RE
060	skewness.mean	DS	Stat	CL+RE
061	skewness.sd	DS	Stat	CL+RE
062	sparsity_mean	DS	Stat	CL+RE
063	sparsity.sd	DS	Stat	CL+RE
064	t_mean.mean	DS	Stat	CL+RE
065	t_mean.sd	DS	Stat	CL+RE
066	var.mean	DS	Stat	CL+RE
067	var.sd	DS	Stat	CL+RE
068	w_lambda	DS	Stat	CL+RE
069	attr_conc.mean	DS	InT	CL+RE
070	attr_conc.sd	DS	InT	CL+RE
071	attr_ent.mean	DS	InT	CL+RE
072	attr_ent.sd	DS	InT	CL+RE
073	class_conc.mean	DS	InT	CL+RE
074	class_conc.sd	DS	InT	CL+RE
075	class_ent	DS	InT	CL+RE
076	eq_num_attr	DS	InT	CL+RE
077	joint_ent.mean	DS	InT	CL+RE
078	joint_ent.sd	DS	InT	CL+RE
079	mut_inf.mean	DS	InT	CL+RE
080	mut_inf.sd	DS	InT	CL+RE
081	ns_ratio	DS	InT	CL+RE
082	best_node.mean	DS	LM	CL+RE
083	best_node.sd	DS	LM	CL+RE
084	elite_nn.mean	DS	LM	CL+RE
085	elite_nn.sd	DS	LM	CL+RE
086	linear_discr.mean	DS	LM	CL+RE
087	linear_discr.sd	DS	LM	CL+RE
088	naive_bayes.mean	DS	LM	CL+RE
089	naive_bayes.sd	DS	LM	CL+RE
090	one_nn.mean	DS	LM	CL+RE
091	one_nn.sd	DS	LM	CL+RE
092	random_node.mean	DS	LM	CL+RE
093	random_node.sd	DS	LM	CL+RE
094	worst_node.mean	DS	LM	CL+RE
095	worst_node.sd	DS	LM	CL+RE
096	leaves	DS	MB	CL
097	leaves_branch.mean	DS	MB	CL
098	leaves_branch.sd	DS	MB	CL
099	leaves_corrob.mean	DS	MB	CL
100	leaves_corrob.sd	DS	MB	CL
101	leaves_homo.mean	DS	MB	CL
102	leaves_homo.sd	DS	MB	CL
103	leaves_per_class.mean	DS	MB	CL
104	leaves_per_class.sd	DS	MB	CL
105	nodes	DS	MB	CL
106	nodes_per_attr.mean	DS	MB	CL
107	nodes_per_attr.sd	DS	MB	CL
108	nodes_per_level.mean	DS	MB	CL
109	nodes_per_level.sd	DS	MB	CL
110	nodes_repeated.mean	DS	MB	CL

Dim Dimension Gr Group LT Learning task DS Data set Stat Statistical
CL Classification RE Regression Int Information Theory LM Landmarking MB Model-based

Table A-24: Meta features database (3/4)

ID	Name	Dim	Gr	LT
111	nodes_repeated.sd	DS	MB	CL
112	tree_depth.mean	DS	MB	CL
113	tree_depth.sd	DS	MB	CL
114	tree_imbalance.mean	DS	MB	CL
115	tree_imbalance.sd	DS	MB	CL
116	tree_shape.mean	DS	MB	CL
117	tree_shape.sd	DS	MB	CL
118	var_importance.mean	DS	MB	CL
119	var_importance.sd	DS	MB	CL+RE
120	algorithm	Alg	Prop	CL+RE
121	alg_type1	Alg	Prop	CL+RE
122	alg_type2	Alg	Prop	CL+RE
123	alg_complexity	Alg	Prop	CL+RE
124	alg_No of hyperparameters	Alg	Prop	CL+RE
125	alg_robust_to_outliers	Alg	Prop	CL+RE
126	alg_robust_to_unscaled data	Alg	Prop	CL+RE
127	alg_robust_to_skewed data	Alg	Prop	CL+RE
128	alg_robust_to_ high dim data	Alg	Prop	CL+RE
129	alg_robust_to_imbalanced data	Alg	Prop	CL+RE
130	alg_robust_to_explainability	Alg	Prop	CL+RE
131	alg_robust_to_comp efficiency	Alg	Prop	CL+RE
132	alg_ave-max	Alg	Perf	CL+RE
133	alg_ave-min	Alg	Perf	CL+RE
134	alg_ave-mean	Alg	Perf	CL+RE
135	alg_ave-stdev	Alg	Perf	CL+RE
136	alg_ave-10_quantile	Alg	Perf	CL+RE
137	alg_ave-25_quantile	Alg	Perf	CL+RE
138	alg_ave-median	Alg	Perf	CL+RE
139	alg_ave-75_quantile	Alg	Perf	CL+RE
140	alg_ave-90_quantile	Alg	Perf	CL+RE
141	alg_ave-95_quantile	Alg	Perf	CL+RE
142	alg_ave-99_quantile	Alg	Perf	CL+RE
143	pip_OneHotEncoderPersonalized	Pip	CB	CL+RE
144	pip_StandardScalerPersonalized	Pip	CB	CL+RE
145	pip_VarianceThresholdPersonalized	Pip	CB	CL+RE
146	pip_RandomUnderSamplerPersonalized	Pip	CB	CL+RE
147	pip_ZScoreSubstitution	Pip	CB	CL+RE
148	pip_PowerTransformer	Pip	CB	CL+RE
149	pip_RemoveCorrelated	Pip	CB	CL+RE
150	pip_SmognPersonalized	Pip	CB	CL+RE
151	pip_MeanImputer	Pip	CB	CL+RE
152	pip_MedianImputer	Pip	CB	CL+RE
153	pip_MinMaxScalerPersonalized	Pip	CB	CL+RE
154	pip_PCAPersonalized	Pip	CB	CL+RE
155	pip_SMOTEENNPersonalized	Pip	CB	CL+RE
156	pip_TargetEncoderPersonalized	Pip	CB	CL+RE
157	pip_RandNoise	Pip	CB	CL+RE
158	pip_SMOTEPersonalized	Pip	CB	CL+RE
159	pip_LastObservedImputer	Pip	CB	CL+RE
160	pip_RFEPersonalized	Pip	CB	CL+RE
161	pip_OrdinalEncoderPersonalized	Pip	CB	CL+RE
162	pip_MV Handling	Pip	Sum	CL+RE
163	pip_Outlier Handling	Pip	Sum	CL+RE

Dim Dimension **Gr** Group **LT** Learning task **DS** Data set **Alg** Algorithm **Pip** Pipeline
CL Classification RE Regression Prop Property Perf Performance CB Method counts top Pip
Sum Sum method counts DPP category

Table A-25: Meta features database (4/4)

ID	Name	Dim	Gr	LT
164	pip_Data Encoding	Pip	Sum	CL+RE
165	pip_Feature Scaling	Pip	Sum	CL+RE
166	pip_Handling Skewness	Pip	Sum	CL+RE
167	pip_Dimensionality Reduction	Pip	Sum	CL+RE
168	pip_Feature Selection	Pip	Sum	CL+RE
169	pip_Augmentation	Pip	Sum	CL+RE
170	pip_Data Balancing (CL/RE)	Pip	Sum	CL+RE
171	pip_Data Cleaning	Pip	Sum	CL+RE
172	pip_Data Transformation	Pip	Sum	CL+RE
173	pip_Data Reduction	Pip	Sum	CL+RE
174	pip_Data Augmentation and Balancing	Pip	Sum	CL+RE
175	pip_max	Pip	Perf	CL+RE
176	pip_min	Pip	Perf	CL+RE
177	pip_mean	Pip	Perf	CL+RE
178	pip_stdev	Pip	Perf	CL+RE
179	pip_10q	Pip	Perf	CL+RE
180	pip_25q	Pip	Perf	CL+RE
181	pip_median	Pip	Perf	CL+RE
182	pip_75q	Pip	Perf	CL+RE
183	pip_90q	Pip	Perf	CL+RE
184	pip_95q	Pip	Perf	CL+RE
185	pip_99q	Pip	Perf	CL+RE

Dim Dimension **Gr** Group **LT** Learning task

CL Classification RE Regression

Sum Sum method counts DPP category

Pip Pipeline Perf Performance

A.13 Meta model

In the following, the feature importances are shown based on which the top 20 and 30 most important meta features are selected to train the meta model once again with a lower amount of features. Three feature importance graphs are exemplarily shown for an *extra trees classifier* (Figure A-22) for classification, *gradient boosting* (Figure A-23) considering all but no pipeline meta features, and *logistic regression* (Figure A-24) by taking into account all meta features available. The *gradient boosting* and *logistic regression* focus on regression.

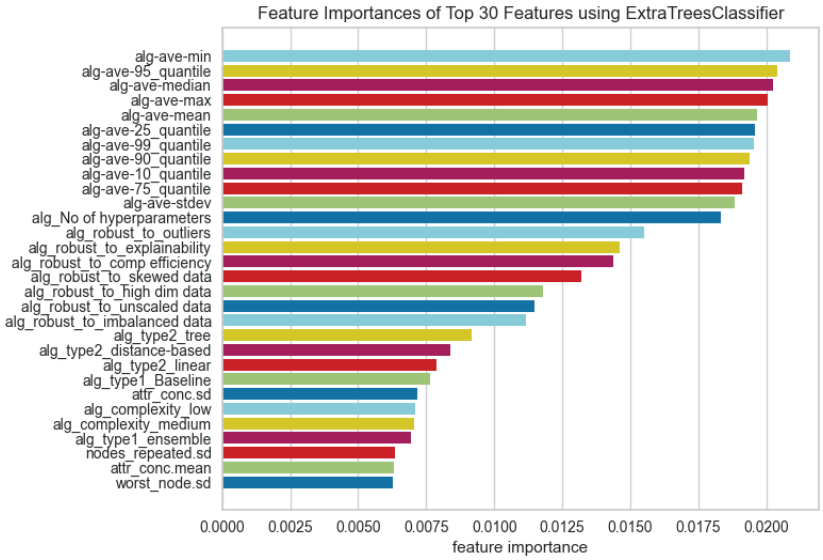


Figure A-22: Top 30 meta features for *extra trees classifier* in case of classification with all but no pipeline-related meta features

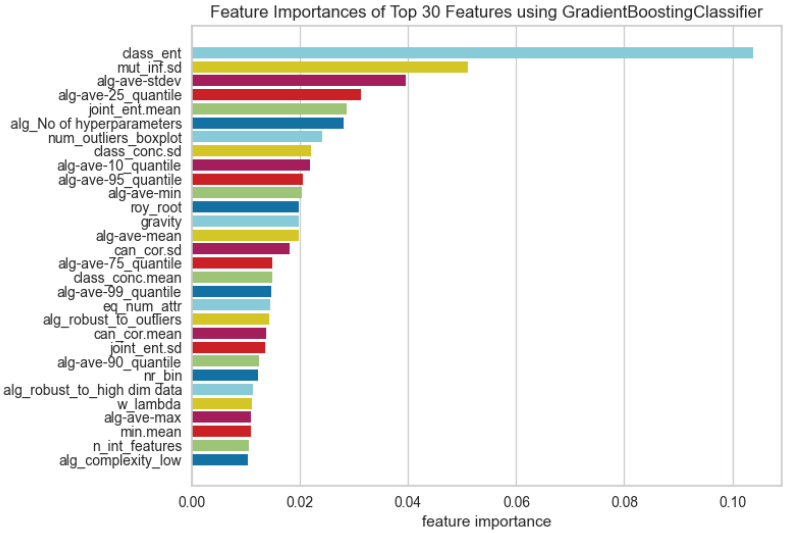


Figure A-23: Top 30 meta features for *gradient boosting classifier* in case of regression with all but no pipeline-related meta features

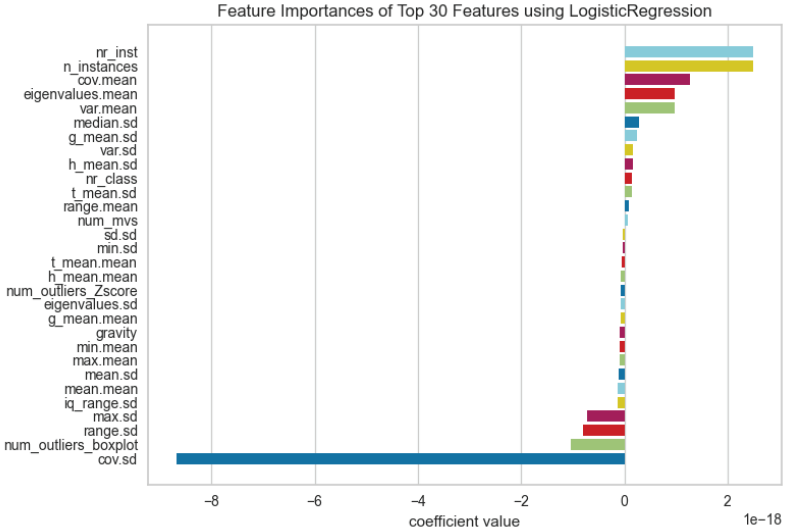


Figure A-24: Top 30 meta features importances for *logistic regression* for regression considering all meta features

A.14 Verification and Validation

The Annex A.14 comprises three major aspects. First, the taxonomy of verification, validation and testing techniques according to BALCI is provided in (I) [BALC98]. The final meta model results according to model performances and DPP scores are shown in (II). Lastly, the results of the execution and special input testing are presented (III).

(I) Verification, validation and testing

Figure A-25 shows the overview of verification, validation and testing techniques according to BALCI that are divided into four different categories.

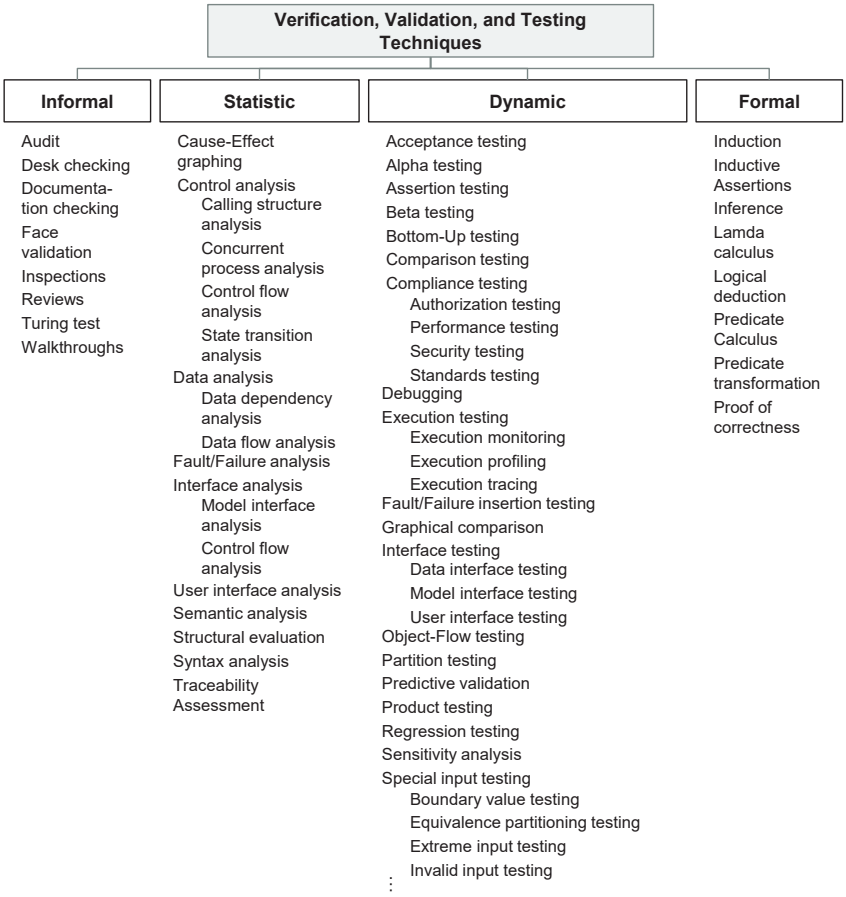


Figure A-25: Taxonomy of verification, validation, and testing techniques (1/2) according to [BALC98]

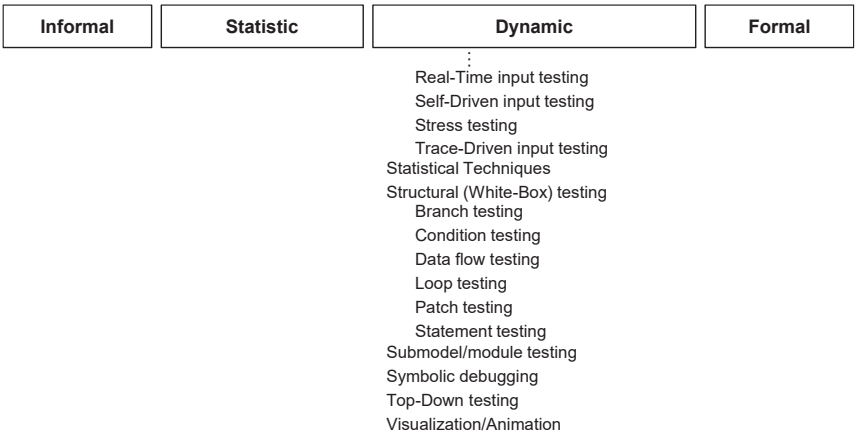


Figure A-26: Taxonomy of verification, validation, and testing techniques (2/2) according to [BALC98]

(II) Meta model performance

The meta model performances for both classification and regression are listed in Table A-26 - Table A-27 and Table A-28, respectively. For every model, the meta feature settings, number of features are provided. The median as well as mean DPP score are given followed by the performance metrics of F1 score, accuracy, precision and recall.

Table A-26: Meta model performances for classification (1/2)

Models	Settings	Features	Median	Mean	F1-Score	Accuracy	Precision	Recall
ET	NoPip-CL	30	0.749	0.720	0.336	0.314	0.342	0.367
GB	NoPip-CL	all	0.739	0.725	0.359	0.349	0.301	0.313
RF	CL	20	0.739	0.722	0.283	0.302	0.301	0.286
GB	NoPip-CL	20	0.736	0.720	0.291	0.325	0.363	0.294
ET	CL	30	0.733	0.719	0.301	0.317	0.323	0.322
GB	NoPip-CL	30	0.731	0.712	0.311	0.397	0.319	0.370
K-NN	CL	20	0.727	0.704	0.256	0.286	0.296	0.311
K-NN	CL	30	0.727	0.704	0.256	0.286	0.296	0.311
K-NN	CL	all	0.727	0.704	0.256	0.286	0.296	0.311
GB	CL	all	0.726	0.720	0.259	0.294	0.296	0.232
ET	NoPip-CL	all	0.720	0.719	0.276	0.294	0.266	0.280
DT	NoPip-CL	30	0.720	0.711	0.270	0.310	0.300	0.360
DT	CL	30	0.717	0.717	0.263	0.373	0.394	0.314

Table A-27: Meta model performances for classification (2/2)

Models	Settings	Features	Median	Mean	F1-Score	Accuracy	Precision	Recall
DT	CL	all	0.714	0.707	0.370	0.262	0.290	0.412
HGBM	NoPip-CL	20	0.713	0.707	0.257	0.286	0.265	0.259
HGBM	NoPip-CL	30	0.713	0.707	0.257	0.286	0.265	0.259
HGBM	NoPip-CL	all	0.713	0.707	0.257	0.286	0.265	0.259
DT	NoPip-CL	all	0.712	0.718	0.291	0.270	0.250	0.311
ET	CL	all	0.710	0.715	0.318	0.317	0.314	0.349
ET	CL	20	0.704	0.704	0.250	0.286	0.259	0.272
GB	CL	30	0.702	0.709	0.257	0.294	0.278	0.262
GB	CL	20	0.701	0.705	0.255	0.294	0.260	0.261
DT	CL	20	0.701	0.704	0.255	0.262	0.283	0.303
ET	NoPip-CL	20	0.697	0.713	0.290	0.317	0.281	0.297

ET Extra trees DT Decision tree HGB Histogram GB GB Gradient boosting K-NN K-nearest neighbor

Table A-28: Meta model performances for regression

Models	Settings	Features	Median	Mean	F-Scores	Accuracy	Precision	Recall
GB	NoPip-RE	all	0.974	0.911	0.310	0.561	0.385	0.333
GB	NoPip-RE	30	0.974	0.911	0.303	0.556	0.380	0.326
GB	RE	all	0.974	0.911	0.282	0.544	0.377	0.310
LogReg	NoPip-RE	20	0.973	0.906	0.280	0.556	0.241	0.333
LogReg	NoPip-RE	30	0.973	0.906	0.280	0.556	0.241	0.333
LogReg	NoPip-RE	all	0.973	0.906	0.280	0.556	0.241	0.333
LogReg	RE	20	0.973	0.906	0.280	0.556	0.241	0.333
LogReg	RE	30	0.973	0.906	0.280	0.556	0.241	0.333
LogReg	RE	all	0.973	0.906	0.280	0.556	0.241	0.333
DT	RE	20	0.972	0.900	0.277	0.500	0.296	0.303
DT	RE	30	0.962	0.893	0.273	0.500	0.367	0.287
GB	RE	20	0.972	0.911	0.272	0.534	0.356	0.303
GB	NoPip-RE	20	0.972	0.908	0.265	0.533	0.381	0.297
GB	RE	30	0.972	0.910	0.254	0.528	0.338	0.291
ET	NoPip-RE	20	0.972	0.910	0.252	0.539	0.301	0.296

ET Extra trees DT Decision tree LogReg Logistic regression GB Gradient boosting

(III) Execution and special input testing

An exemplary result and proof of functionality during execution testing can be taken from Figure A-27. On the left, the user interface for classification is given, while on the right, the regression interface is shown.

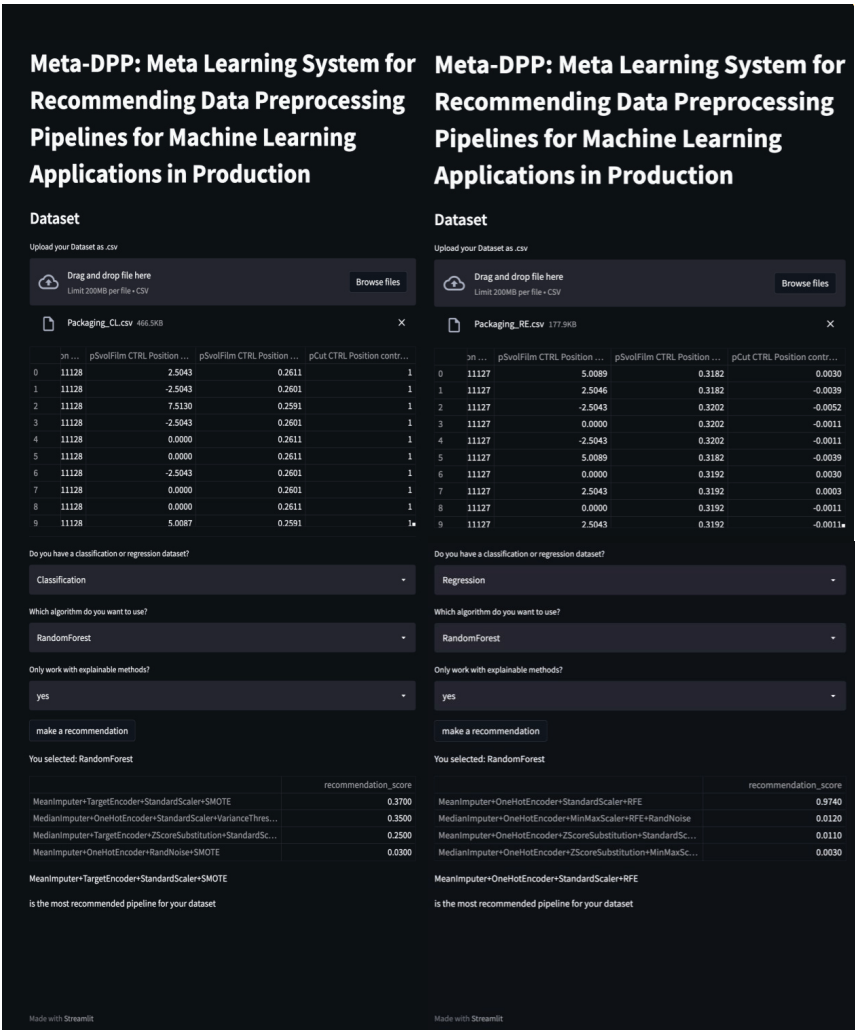


Figure A-27: User interface and exemplary results of execution testing for both classification and regression

Besides the execution testing, missing input testing was performed. In case of missing inputs, Meta-DPP provides a recommendation, since the ML algorithm and explainable methods are filled by default. Regarding the wrong inputs, the results in case a classification data set is inserted but regression task selected can be seen on the left of Figure A-28. On the right of Figure A-28, the regression data set is inputted but learning task classification selected. As it can be seen in the middle, a warning is provided but Meta-DPP still is applicable.

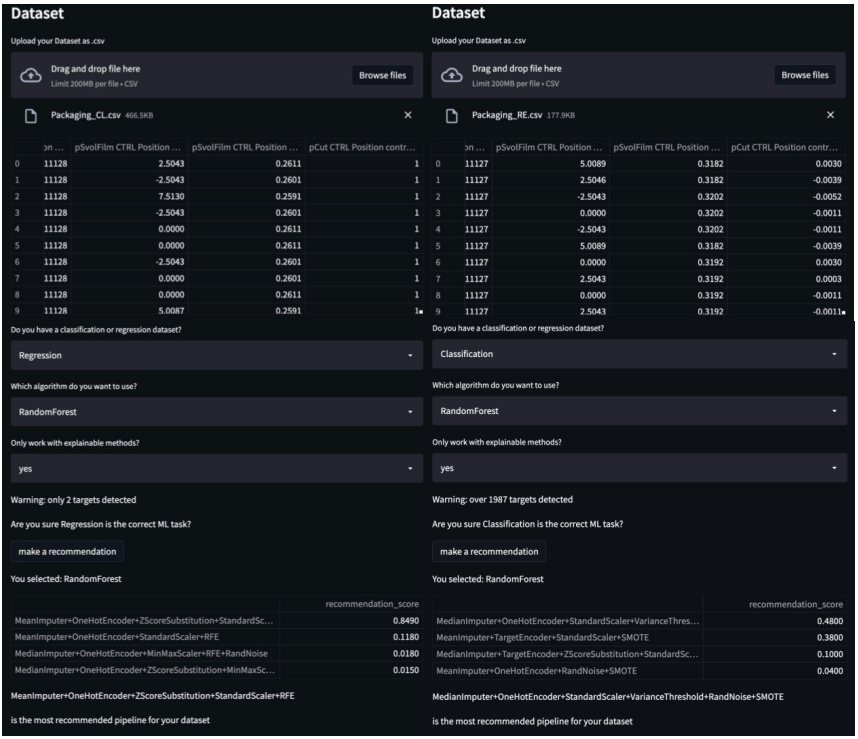


Figure A-28: Wrong input testing results (Wrong Input 1 left & Wrong Input 2 right)

In case a wrong data set is inserted, the warning arises that the wrong data set is inserted. Besides that, two tests have been performed to identify if the feature regarding the explainability works as intended. Figure A-29 shows the results of the explainability input testing. On the left, the explainability is required, while on the right, the explainability is provided as no.

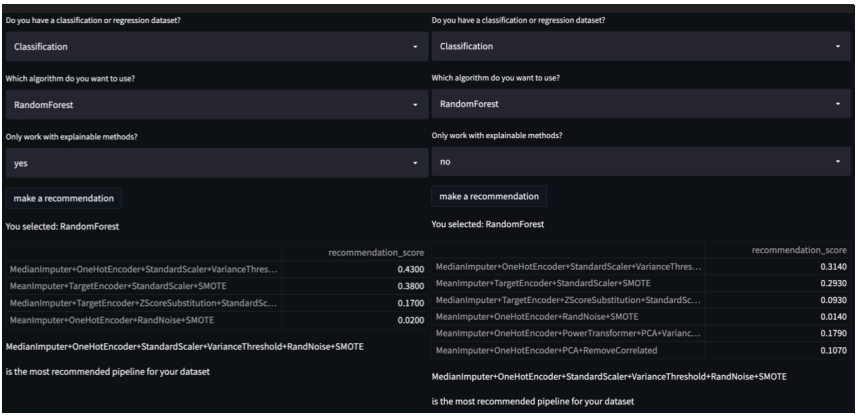


Figure A-29: Explainability input testing results: explainability yes (left), no (right)

B.1 Code for Benchmarking

The code can be found in:
https://gitlab.cc-asp.fraunhofer.de/IPT_300/datapreprocessingandml/dpp-pipelines

B.2 Code for Meta-DPP

The code can be found in:
https://gitlab.cc-asp.fraunhofer.de/IPT_300/datapreprocessingandml/dpp-pipelines