Data driven modeling & optimization of industrial processes

Stein, B. van

Data Driven Modeling & Optimization of Industrial Processes

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus prof.mr. C.J.J.M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op donderdag 20 september 2018
klokke 13.45 uur

door

Bas van Stein

geboren te Sassenheim, Nederland
in 1989

Promotiecommissie

Promotor:	Prof. Dr. T.H.W. Bäck	
Co-promotor:	Dr. W.J. Kowalczyk	
Overige leden:	Prof. Dr. A. Plaat	
	Prof. Dr. F.J. Verbeek	
	Prof. Dr. H.H. Hoos	
	Dr. H.C.M. Kleijn	
	Prof. Dr. S. Manegold	(CWI, The Netherlands)
	Prof. Dr. B. Filipic	(Jozef Stefan Institute, Slovenia)
	Prof. Dr. J. Mehnen	(University of Strathclyde, UK)

This research is financially supported by the Dutch funding agency NWO, under project number 650.002.001 (the PROMIMOOC project), in collaboration with Tata Steel IJmuiden, BMW Group Regensburg, Centrum voor Wiskunde en Informatica (CWI) and MonetDB.

Printed by: ProefschriftMaken || DigiForce

Contents

1 Introduction **1**
1.1 Background 1
1.2 Objectives 5
1.3 Outline 6

Author's Contributions **8**

2 The PROMIMOOC Project **11**
2.1 Tata Steel 11
2.1.1 Objectives 12
2.1.2 Data 13
2.2 BMW 14
2.2.1 Objectives 14
2.2.2 Data 15
2.3 A Generic Framework for Data driven On-line Control 15

3 Missing Value Analysis and Imputation **19**
3.1 Introduction 20
3.2 Missing Value Analysis 22
3.2.1 Missing Data Types 22
3.2.2 Patterns of Missing Values 23
3.2.3 Analyzing Missing Value Patterns 26
3.2.4 Analysis of Existing Data Sets 28

3.3 Incremental Attribute Regression Imputation 30
3.3.1 Existing Imputation Algorithms 32
3.3.2 Experimental Setup 35
3.3.3 Results 36
3.4 Attribute Selection and Sorting Methods 41
3.4.1 Greedy Model Accuracy Selection 41
3.4.2 Greedy Imputation Quality Selection 43
3.5 Conclusions 43

4 Outlier Detection in High-Dimensional Big Data 47
4.1 Introduction 49
4.1.1 Related Work 52
4.2 Global Local Outliers in SubSpaces 53
4.2.1 Problem Definition 53
4.2.2 Preliminaries 56
4.2.3 Global Local Outlier Probabilities 58
4.2.4 Subspace Search 59
4.3 Experiments 60
4.3.1 Synthetic Data 61
4.3.2 Benchmark Data with Implanted Outliers 62
4.3.3 Benchmark Data with Minority Class as Outliers 65
4.4 Case Study: Outlier Detection for BMW 65
4.5 Conclusions and Outlook 70

5 Cluster Kriging 73
5.1 Introduction 75
5.2 Kriging 76
5.3 Relevant Research 78
5.4 Cluster Kriging 82
5.4.1 Clustering 82
5.4.2 Modeling 86
5.4.3 Prediction 86
5.5 Flavors of Cluster Kriging 91
5.6 Experimental Setup and Results 92

5.6.1 Hyper-Parameters . . . 93
5.6.2 Quality Measurements . . . 94
5.6.3 Results . . . 95
5.6.4 Parameter Setting Recommendations . . . 96
5.7 Efficient Global Optimization . . . 98
5.7.1 The Efficient Global Optimization Algorithm . . . 100
5.7.2 Cluster Kriging-based EGO . . . 101
5.7.3 Experiments . . . 102
5.8 Conclusions . . . 105

6 Arbitrary Model Efficient Global Optimization 109
6.1 Background . . . 111
6.2 kNN Uncertainty Measure for EGO . . . 111
6.3 Experimental Setup . . . 113
6.4 Conclusion . . . 117

7 Conclusions and Outlook 121
7.1 Conclusions . . . 122
7.2 Future work . . . 124

A IARI results 126

B Symbols and Abbreviations 137

English Summary 151

Nederlandse Samenvatting 154

About the Author 157

1 Introduction

1.1 Background

Industry has shown a lot of interest over the last decades in the emerging fields of (big) data mining, machine learning, and deep learning. Industry sees opportunities to optimize their production processes and further automate their process pipeline with the help of data driven technologies. With the data that has been collected in these industrial processes for the past decade, predictive models can be trained and data driven optimization of these processes becomes feasible. For industry it is crucial to stay ahead of the competition by producing good quality products in as little time as possible and by spending a minimum of resources, keeping the costs low. Data mining, machine learning and model-based optimization are important techniques that industry can benefit from by automatically controlling and optimizing their complex production processes.

Optimization and feedback loops have always played an important role in the manufacturing industry. Often, these feedback loops are restricted to one part of the process or to one particular machine, due to the complexity of modeling the interactions between the different process steps and external factors that are hard to model. These feedback loops are most often reacting to the current state given a predefined preferred state, like a central heating system that maintains a room temperature of twenty degrees Celsius by turning on and off the heater, given that the current temperature is either too low or too high. Current implementations of more complex optimization procedures are mainly focused on

optimizing unit processes and are often implemented using complex mathematical models that require a lot of domain knowledge and expertise. In the field of *Optimal Operational Control* [1, 2, 3, 4] and *Real Time Optimization* (RTO) [5, 6, 7] static mathematical models are constructed of the industrial process and used in the optimization procedure to come up with good operational parameters for a specific unit process. These model-based control theories include both linear and nonlinear systems where preferred parameters of the controllers are assumed to be known. These procedures are static and often not able to learn from historical events or to adapt when the process or environment changes.

With the current state of the art technologies, infrastructure and collected big data sets from sensors and systems, new possibilities arise to optimize more challenging industrial processes in a scalable and robust way. These technologies allow to model and optimize not just one part of the production process, but the complete process. Enabling us to capture the complex interactions of the different stages and predicting defects and possible defect causes much earlier. Data driven modeling also allows for adapting to concept shift and concept drift, as models can be easily retrained or even adapted in an on-line fashion. With big data, the possibility to build predictive models purely on the data becomes feasible. With these predictive models, optimization algorithms can produce a set of optimized parameters to improve an entire production chain on multiple optimization criteria, delivering both optimized process parameters as well as additional insight into the production process itself by analyzing the correlations and trade-offs between different objectives.

However, the data driven modeling and optimization of these complex processes require insight in the data, domain expertise and a complete pipeline of data acquisition, data cleansing, data preprocessing, modeling, post-processing, validation and optimization. Each of these pipeline stages come with different challenges and possible solutions. In this dissertation a framework for these different stages is proposed, specifically tailored for the industrial processes of steel-making and car body manufacturing. Throughout this dissertation, many of these stages are addressed, in particular the preprocessing, modeling, validation and optimization stages. The following research questions are highly relevant to these stages and are answered in this thesis:

- How can supervised and unsupervised machine learning techniques be utilized in the context of complex industrial processes?
- How can real-world data issues, such as missing data, be treated efficiently?
- How can unsupervised learning be used for anomaly detection in high-dimensional industrial applications?
- How can predictive models such as Kriging be efficiently applied to a large amount of data?
- How can global optimization be applied in an industrial context, even in real-time?

To answer these questions, several novel algorithmic contributions are presented in this thesis, and explained in detail. The proposed framework and algorithmic contributions are applied on two really challenging industrial manufacturing processes, steel-making and car body manufacturing, in the context of the PROMIMOOC project, "PROcess MIning for Multi-Objective Online Control" [1]. This project is executed in collaboration with three industrial partners, Tata Steel, BMW Group and MonetDB and in collaboration with CWI (Centrum Wiskunde & Informatica). The two industrial processes that are discussed in this thesis are the manufacturing of car body parts at BMW Group located in Regensburg, Germany, and steel-making at Tata Steel located in IJmuiden, The Netherlands.

Tata Steel IJmuiden

Tata Steel IJmuiden is a part of Tata Steel Europe, which in turn belongs to the multi-national company Tata Group. More than 9.000 people work at Tata Steel IJmuiden, producing more than 7 million tonnes of high quality steel. With over 750 hectares of company terrain (Figure 1.1a), Tata Steel IJmuiden is the largest company in the Netherlands. During the process of Hot Rolling, which is only one of the process steps required to make high quality steel, more than 6000 signals are collected roughly every 10 milliseconds, producing approximately $2 \cdot 10^{13}$ data

[1] PROMIMOOC project (project number: 650.002.001), funded by NWO (Netherlands Organisation for Scientific Research).

(a) Tata Steel IJmuiden. (Photo from NU.nl)

(b) Hot Strip Mill 2. (Photo from viktormacha.com)

Figure 1.1: Images from Tata Steel IJmuiden.

points per year, resulting in more than 30 TB of data, not even considering the images and video material stored as well.

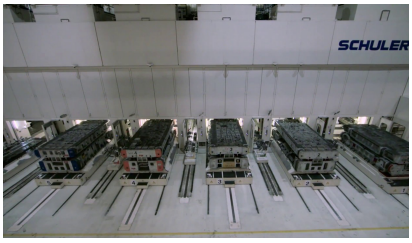
As can be seen in Figure 1.1b, the Hot Rolling process is a very rough process with a lot of mechanical parts, very high temperatures and many internal and external factors that can influence the results.

BMW Regensburg

BMW Group Regensburg (Figure 1.2a) is one of BMWs production plants where they produce eight different models on a single production line. The press shop (Figure 1.2b) is the part of the production line where car body parts, such as side frames and roof tops, are produced by stamping steel blanks. The press line is 54 meters long, weights about 4.500 tonnes and delivers 8.100 tonnes of pressure using five press stations. The press line consumes on average 150 tonnes of steel per day and produces more than 4.2 million parts per year. In comparison with the hot rolling process of Tata Steel, the BMW press shop is a very controllable and clean process. However, many machine parameters and several external factors, such as temperature, influence the press line. Currently, most of the machine parameters are set by domain experts.

While the process of hot-rolling steel is completely different from the process of stamping car body parts, both processes can be modeled and optimized using

(a) BMW Group Regensburg (Photo from autointell-news.com)



(b) BMW Press Shop Regensburg. (Photo from autointell-news.com)

Figure 1.2: Images from BMW Group Regensburg.

similar techniques and the same generic data driven framework.

1.2 Objectives

In the car body parts manufacturing and steel industry, data mining and online automated quality control are emerging and important topics [8, 9]. In an *Industry 4.0* factory, machines and products are interlinked with each other as one collaborative process. The PROMIMOOC project anticipates on this idea of a completely automated and self-optimizing production chain. The main objective of the PROMIMOOC project is to develop a generic data driven platform for data collection, integration, modeling and model-based online process control. In this data driven platform the industrial production process can be monitored, optimized and adapted in real-time.

Such a generic platform consists of several main components:

Extraction Extraction, transferring and loading (ETL) of the data: This component deals with the extraction of the machine measurements and settings, aligning the different measurements with quality indicators and storing the data in a fast column-store database.

Preprocessing Data preprocessing, feature extraction and feature selection: The second component deals with the cleaning, preprocessing and extracting informative features from the data.

Exploration Exploratory Data Mining and unsupervised learning: In the third component, exploratory data mining techniques are used to gain insight into the different features and different types of available data. This component also deals with finding anomalies in the data, detecting anomalous events and finding clusters of data points.

Prediction Predictive model development and maintenance: Component four is about training, validating and optimizing data driven predictive models in order to predict various process indicators such as cost and product quality from material input measurements and machine parameters.

Optimization Model-based multi-objective optimization: The last component is using the predictive models to perform multi-objective optimization in order to improve the various process indicators by giving real-time suggestions of machine parameters.

The research discussed in this thesis aims at the second, third, fourth and fifth component, preprocessing, feature extraction, feature selection, exploratory data mining, predictive modeling and optimization. The first component of the framework, *Extraction*, is outside the scope of this research as the ETL process is handled by MonetDB [10].

1.3 Outline

In the following chapter, a detailed overview of the PROMIMOOC project and the industrial partners, BMW and Tata Steel, is given. A generic framework to perform anomaly detection, predictive modeling and optimization for industrial processes is presented in Section 2.3. The first component of the framework, preprocessing, is discussed in Chapter 3, where one of the main issues in preprocessing data sets, missing values, is discussed in detail. In addition, several techniques are proposed to visualize and impute (repair) these missing values. Unsupervised

learning and more specifically, anomaly detection, is discussed in Chapter 4. An anomaly detection algorithm that works for high-dimensional mixed data sets is presented and empirically evaluated using both synthetic and real-world data sets. In Chapter 5, a novel Kriging approximation technique, *Cluster Kriging*, is presented with various algorithmic flavors that allow the use of Kriging for much bigger data sets by reducing the time and space complexity. The use of Cluster Kriging in Efficient Global Optimization is discussed and evaluated to show that it is feasible to apply model-based global optimization to big data. To allow the use of other predictive models in combination with Efficient Global Optimization, a heuristic uncertainty measure is proposed in Chapter 6. With the help of this heuristic it becomes possible to use deep artificial neural networks trained on the PROMIMOOC data sets in combination with the Efficient Global Optimization algorithm to perform optimization in near real-time for industrial production processes. Finally, conclusions and future work are discussed in Chapter 7.

Author's Contributions

[1] Bas van Stein et al. "Optimally weighted cluster kriging for big data regression". In: *International Symposium on Intelligent Data Analysis*. Springer, Cham. 2015, pp. 310–321. DOI: 10.1007/978-3-319-24465-5_27.

[2] Bas van Stein, Wojtek Kowalczyk, and Thomas Bäck. "Analysis and Visualization of Missing Value Patterns". In: *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*. Springer International Publishing. 2016, pp. 187–198. DOI: 10.1007/978-3-319-40581-0_16.

[3] Bas van Stein and Wojtek Kowalczyk. "An Incremental Algorithm for Repairing Training Sets with Missing Values". In: *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*. Springer International Publishing. 2016, pp. 175–186. DOI: 10.1007/978-3-319-40581-0_15.

[4] Bas van Stein, Matthijs van Leeuwen, and Thomas Bäck. "Local subspace-based outlier detection using global neighbourhoods". In: *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE. 2016, pp. 1136–1142. DOI: 10.1109/bigdata.2016.7840717.

[5] Pepijn van Heiningen, Bas van Stein, and Thomas Bäck. "A framework for evaluating meta-models for simulation-based optimisation". In: *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*. IEEE. 2016, pp. 1–8. DOI: 10.1109/ssci.2016.7850207.

[6] Bas van Stein et al. "Fuzzy clustering for optimally weighted cluster kriging". In: *Fuzzy Systems (FUZZ-IEEE), 2016 IEEE International Confer-*

ence on. IEEE. 2016, pp. 939–945. DOI: 10.1109/fuzz-ieee.2016.7737789.

[7] Bas van Stein et al. "Towards Data Driven Process Control in Manufacturing Car Body Parts". In: *Computational Science and Computational Intelligence (CSCI), 2016 International Conference on*. IEEE. 2016, pp. 459–462. DOI: 10.1109/csci.2016.0093.

[8] Bas van Stein et al. "Cluster-based Kriging Approximation Algorithms for Complexity Reduction". In: *Data Mining and Knowledge Discovery* (2018), Under review. DOI: 10.1145/3071178.3071321.

[9] Sander van Rijn et al. "Algorithm configuration data mining for CMA evolution strategies". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2017, pp. 737–744. DOI: 10.1145/3071178.3071205.

[10] Hao Wang et al. "Time complexity reduction in efficient global optimization using cluster kriging". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2017, pp. 889–896. DOI: 10.1145/3071178.3071321.

[11] Hao Wang et al. "A new acquisition function for Bayesian optimization based on the moment-generating function". In: *Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on*. IEEE. 2017, pp. 507–512. DOI: 10.1109/smc.2017.8122656.

[12] Bas van Stein et al. "A Novel Uncertainty Quantification Method for Efficient Global Optimization". In: *Information Processing and Management of Uncertainty in Knowledge-Based Systems. Theory and Foundations*. Springer International Publishing. 2018, forthcoming. ISBN: 978-3-319-91475-6. DOI: 10.1007/978-3-319-91476-3.

[13] Roy de Winter et al. "Designing Ships using Constrained Multi-Objective Efficient Global Optimization". In: *Machine Learning, Optimization, and Data Science*. Springer. 2018, forthcoming.

2 The PROMIMOOC Project

In the PROMIMOOC project, the two real-world use cases available are the production of steel coils (Tata Steel) and the stamping of car body parts (BMW). Together these two cases nicely reflect a complete industrial process, where we have a producer of steel coils on one hand and a consumer of these (and other) steel coils that in turn produces car body parts on the other hand. In this chapter both cases are explained in detail. In addition, a generic framework for data driven on-line control that can be applied to many of these industrial processes is proposed in Section 2.3.

2.1 Tata Steel

In the steel-making industry, iron ore and scrap are transformed into smooth steel coils of a few centimeters thick and many meters in length. The process consists of several steps, some of the steps being optional and some steps are sometimes executed multiple times.

Continuous Casting Liquid iron from the blast furnace is being cast into thick heavy slabs.

Hot rolling The slabs are being reheated by a walking beam or pusher furnace and go through several rougher and finishing mills. Each mill is reducing the thickness of the steel and increasing its length.

Pickling In the pickling line, the coils get cleaned.

Cold rolling After hot rolling, most coils get cold rolled by several more reduction mills to get the dimensions the customer requires.

Galvanizer Some of the steel coils need a coating and further processing, this is what the galvanizer is for.

For the PROMIMOOC project, only the hot rolling process step of Tata Steel is taken into account since this is the step where surface defects start to occur and where machine parameters have a high impact on the final product. The hot rolling process can by itself be divided into a dozen smaller steps as can be observed in Figure 2.1. It consists of four furnaces, two walking beam furnaces and two pusher furnaces. Each steel slab will pass from one of these furnaces and will then go through five consequent rougher mills. After the rougher mills a cropper shear is used to remove any oxide from the steel surface. The steel then passes another seven finishing mills before it ends up at the roll-out table. At the roll-out table the steel cools down before it is being coiled by the coiler. At the roll-out table the surface inspection system takes images of each millimeter of steel, both of the upper and lower surface, and detects and classifies defects on the surface. These defects are classified into twenty-seven defect families.

2.1.1 Objectives

For Tata Steel, the main objectives are to accurately classify defects on the surface of the steel coils by using material measurements and machine parameters. Finding relations, possible causes and anomalies in the provided data is of great importance as this brings additional insight into the complex process of hot rolling and may lead to an improved production process.

Once surface defects can be classified and predicted using input material properties and machine parameters, model-based optimization of the machine parameters can be performed using optimization algorithms. These algorithms can give recommendations of near-optimal machine settings that can then be used by a domain expert in controlling the production process.

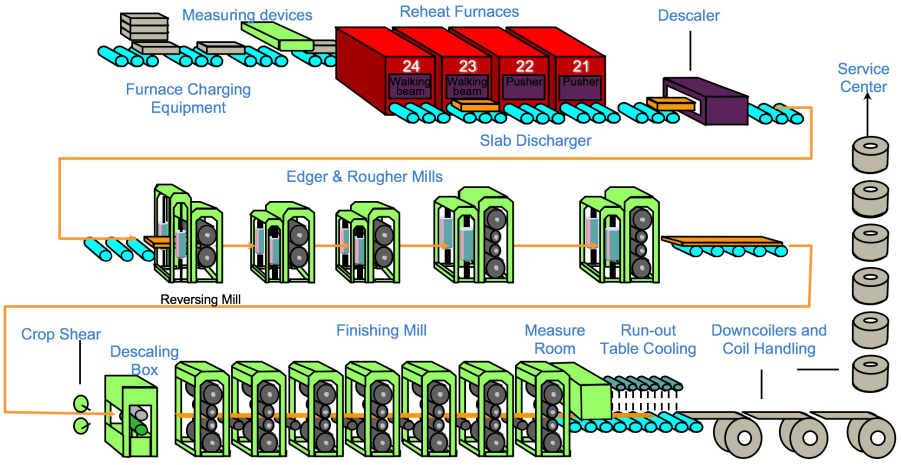


Figure 2.1: A schematic view of the hot rolling process at Tata Steel's hot strip mill 2 (HSM 2). Courtesy of Tata Steel.

2.1.2 Data

The data sets provided by Tata Steel contain measurements, machine parameters and defect information for roughly 20.000 steel coils processed by Hot Strip Mill 2. The data is divided over numerous tables, the most important sets are the Rougher Mill (RM), Finishing Mill 1 (FM1), Finishing Mill 2 (FM2) and Defect data sets. These four data sets have different sampling rates and are therefore not trivial to combine. Using timestamps, relative positions and additional meta-data, the four data sets have been combined in data views by CWI in a MonetDB database environment. There are several thousand records available per coil. Each record is in turn consisting of up to hundred signals that were used for the experiments in this research.

2.2 BMW

In the car body parts industry, blanks of sheet metal are cut from a coil and pressed into car body parts such as side frames, roofs and structural parts like B-pillars. For different parts, different material is required and different machine settings can be used. Due to the high variation as well as high-dimensionality in both material properties and machine settings, the process is a very complex one with lots of parameters that influence the final product.

The manufacturing process consists of two main process steps and a buffer period. First, the incoming steel coils are unrolled and cut into individual blanks. The steel blanks are then stacked on top of each other and stored in the buffer. After a certain time in the buffer, the stack of blanks are moved to the press line. At the press line the blanks are pressed into a specific car body part. Depending on the body part produced, the press line consists of a number of operations, each of them controlled by a large variety of machine parameters.

2.2.1 Objectives

On-line quality optimization of the products and the prediction and avoidance of defects are the key goals of this research for BMW. More precisely, the aim is to estimate the occurrence of defects and to warn domain experts of incoming anomalously looking material and abrupt changes in material flow such that machine parameters can be adjusted in time.

To estimate the occurrence of defects, data mining techniques have to be applied at the very beginning of the production process. Anomaly detection [11] plays an important role in this early stage, since most of the machine parameters are still unknown. Using anomaly detection techniques on material properties allows for the detection of anomalous metal coils and more precisely, regions in the sheet metal that could later lead to problems in the production process. The results of anomaly detection algorithms can be presented to experts to gain additional knowledge about the process and to warn the press line controllers of risks as early as possible. However, to apply anomaly detection and other unsupervised

techniques to the BMW data, some challenges have to be tackled. The dimensionality of the problem is large and the data consists of heterogeneous coil types and suppliers used for many different car body parts. Not all coil measurements are annotated with a supplier and final product type which makes it difficult to split the data set.

2.2.2 Data

Most of the BMW data comes from the first production step, the cutting process. At the cutting process, the following properties are measured over the complete coil length.

Impulse Magnetic Process On-line Controller (IMPOC) is an advanced measurement commonly used in steel manufacturing plants that measures the residual magnetic field strength of the material [12].

Oil Levels on the surface of the blanks are considered to be an important factor in the stamping process. The amount of lubricant affects the friction and thus plays an important role in the deep drawing process of sheet metals.

Roughness of the surface.

Thickness of the material.

Peak Count of the surface, representing the number of peaks per square meter.

The oil levels are measured by a sensor that moves over the width of the coil, all other sensors are placed at the center of the cutting machine. Additional machine parameters such as re-oiling and six cylinder forces used in the stamping process are stored and linked to the steel blanks in the database.

2.3 A Generic Framework for Data driven On-line Control

The optimization of these processes is far from trivial and though they have many objectives in common, the two processes are composed of different steps, machines

and data. A generic framework for data driven optimization of process parameters and on-line control is proposed as a solution to this problem. Each step required for the framework to work, as already given in the Introduction (Section 1.2), is covered by the proposed framework.

Both processes are so called semi-batch processes, where the products are produced in a batch fashion but the production of each individual product can be seen as a continuous process. For example, while several steel coils can be seen as a batch, the production of one steel coil is the continuous casting and reduction of roughly 2000 meters of steel. Due to the semi-batch nature of these processes, the generic framework consists of steps focused on batch processes, such as the prediction of good machine parameters for the manufacturing of the next products, and continuous processes, such as the detection of anomalous regions in the input material. The framework also has to deal with high-dimensional data coming in real-time or close to real-time. The framework needs to provide valuable feedback to the domain experts, decision makers and process controllers in limited time, about the current and possible future situation of the production process. A schematic overview of the proposed framework is shown in Figure 2.2.

The framework consists of the as-is production process on the left, where the actual production process is abstracted to one step for the sake of simplicity. The data gathered by the production process consists of three types, material measurements, process parameters and product quality measurements. All three data types are required for most of the data driven framework modules to work. In case quality measurements are absent, supervised predictive modeling would not be possible but anomaly detection and monitoring would be still fine. The first step of the data driven framework is to preprocess and clean the incoming material measurements and planned process parameters. The preprocessed data is stored in a fast database (in our setup MonetDB). Using the input data, anomaly detection and unsupervised algorithms can provide the operator with valuable insights even before the production takes place. When a trained predictive model is available, the input data can also be used to provide the operator with suggestions of near-optimal process parameters using model-driven optimization techniques. Once the product is produced, a quality inspection system can provide feedback to the operator. This data can also be used to perform supervised learning and train or

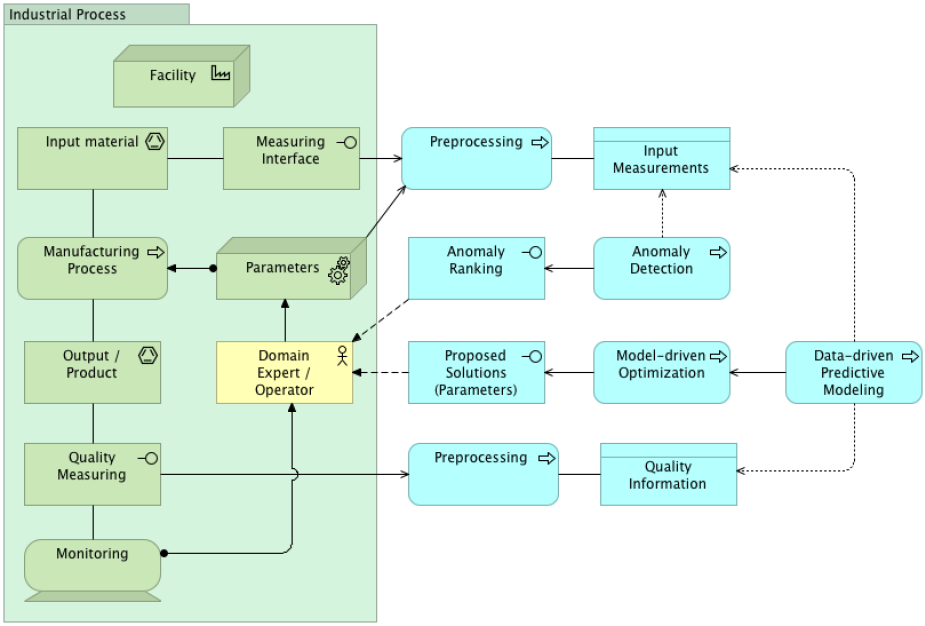


Figure 2.2: A data driven framework for optimizing and monitoring manufacturing processes.

re-train the data driven predictive models for the next iteration of the production process.

The next chapters present solutions and implementations of each step of the proposed framework. In each chapter a reference to the relevant framework step is given.

3 Missing Value Analysis and Imputation

In the first steps of the data driven framework, data-preprocessing is of high importance. Many issues can occur in real-world applications such as erroneous data, missing data, noise, cumbersome code, alignment issues between different devices, encodings and many more. Here we focus on the problem of missing values. Missing values play an important role in the data-preprocessing step as they negatively influence the usability of the data and the results of many data driven algorithms. Dealing correctly with these missing or even erroneous values is not a trivial task. Naively removing all records with missing values will lead to bias in the data and a loss of information, while repairing or replacing the missing values might also result in bias, erroneous values and mis-information. In this chapter several techniques of visualizing missing values are proposed to gain a better understanding of the missing value mechanisms. In addition an algorithm is proposed to effectively repair missing values in an iterative column-wise fashion.

In this chapter the challenges of missing values are examined and for both the exploration and imputation of these missing values, a solution is proposed. The content of this chapter is primarily based on the two publications [13] and [14].

3.1 Introduction

In industrial processes and many other real-world applications, data points are collected to gain insight into the process and to make important decisions. Understanding and making predictions for these processes are vital for their optimization. Missing values in the collected data cause additional problems in building predictive models and applying them to fresh data. Unfortunately, missing values are very common and occur in many processes, for example, sensors that collect data from a production line may fail; a physician that examines a patient might skip some tests; questionnaires used in market surveys often contain unanswered questions, etc. This problem leads to the following questions:

1. What are the causes for missing values and can patterns of missing values be observed?
2. How to build high quality models for classification and regression, when some values in the training set are missing?

Gaining more insights into the patterns of the missing values is an important factor for selecting algorithms that are appropriate for a given data set. A theory about missing value patterns and mechanisms [15, 16, 17, 18, 19] already exists, but the existing theory is insufficient to gain a clear understanding of each possible pattern of missing values because it only defines a small set of possibilities.

Proposed is an extension to the current theory, covering all patterns of missing values occurring in a data set, ranging from a completely *Univariate Pattern* to a completely *Arbitrary Pattern*. Using this new concept, a greedy algorithm that analyzes data sets is proposed, together with various visualization techniques that provide a clear overview of the patterns of missing values occurring in the data set. Besides analyzing missing values, it is also interesting that the same techniques can be used to analyze the patterns of occurrence of a specific value. For example, the patterns in sparse data sets (where 0 would be the unique value to analyze, or even more interesting where all non-zero values are analyzed).

After analyzing the missing value patterns, these missing values need to be dealt with in such a way that predictive models can be fitted on the data set. There

are several methods developed for tackling this imputation problem, see e.g., [20, 21, 16, 22, 23]. The most common method, *imputation*, reconstructs the missing values with help of various estimates such as means, medians, or simple regression models which predict the missing values. Proposed is a more sophisticated approach, *Incremental Attribute Regression Imputation* (IARI) which prioritizes all attributes with missing values and then iteratively "repairs" each of them, one by one, using values of all attributes that have no missing values or are already repaired, as predictors. Additionally, the target variable is also used as a predictor in the repair process. Repairing an attribute is achieved by constructing a regression model and applying it for estimation of missing values. The Random Forest algorithm, [24], [18], is used in these experiments due to its accuracy, robustness, and versatility: it can be used to model both numerical and categorical variables. Obviously, after repairing all attributes with missing values a final model for the original target variable can be trained on the repaired training set.

The proposed algorithm is evaluated using five well-known data sets: *Digits*,*Page Blocks*,*Concrete*, and *CoverType* from the UCI Machine Learning Repository, [25], and *Housing 16H* from mldata.org [26], first removing some values at random, and then reconstructing them with help of IARI and several common imputation algorithms. Finally the quality of these imputation methods is compared by measuring the accuracy of regression and classification models trained on the reconstructed data sets. The results demonstrate that in most cases, no matter how many attributes were spoiled and how severely, the IARI algorithm outperformed other imputation methods both in terms of the accuracy of the final models and the accuracy of imputation. On the other hand, the IARI algorithm is computationally very demanding–it builds as many Random Forests as the number of attributes that should be repaired. Fortunately, due to the parallel nature of the Random Forest algorithm, the runtime of the IARI algorithm can be easily reduced by running it on a system with multiple cores or CPUs.

This chapter first elaborates on possible patterns and causes of missing values in data sets and how to analyze and visualize these patterns. Later on in the chapter the IARI algorithm is explained in detail and results from various experiments are presented and discussed.

3.2 Missing Value Analysis

In the following two subsections an overview of common definitions of several mechanisms behind missing data is given, and several new concepts of "patterns of missingness" are introduced.

3.2.1 Missing Data Types

Rubin et al [15] defined three major classes of missing values: *Missing Completely At Random* (MCAR), *Missing At Random* (MAR), and *Missing Not At Random* (MNAR). Informally, we say that values in a data set are Missing Completely At Random (MCAR), if the probability distribution of "being missing" is completely independent on the observed or missing data. When the probability distribution of "being missing" somehow depends on the observed (non-missing) values, then we talk about the Missing At Random (MAR) scenario. Finally, when "being missing" depends on the actual, unobserved values, then we talk about the Missing Not At Random (MNAR) scenario.

To illustrate these three definitions, let us consider data of patients collected in a hospital. When a doctor decides not to measure patient's body temperature because she can already see that the temperature is too high, then we have the MNAR scenario - the decision of not measuring the parameter depends on its actual value. On the other hand, if the temperature is systematically measured, but from time to time the data registration process malfunctions (independently on the measured values), then we have the MCAR scenario. Finally, if the doctor has a habit of not measuring the temperature of patients with high blood pressure (and blood pressure is always registered), then we have a MAR scenario.

Formally, the three scenarios can be summarized in the following definition, [18]:

Definition 3.1 *Let y denote a target attribute, X a matrix of input attributes with missing values, X_{obs} the observed entries in X, $Z = (y, X)$, $Z_{obs} = (y, X_{obs})$. Additionally, let R denote an indicator matrix with ijth entry 1 if x_{ij} is missing and 0 otherwise.*

We say that the data is Missing Completely At Random if:

$$Pr(R|Z, \theta) = Pr(R|\theta).$$

We say that the data is Missing At Random if:

$$Pr(R|Z, \theta) = Pr(R|Z_{obs}, \theta).$$

We say that the data is Missing Not At Random if it is not Missing at Random. (Here we assume that probability distributions are parametrized by parameters θ.)

3.2.2 Patterns of Missing Values

In addition to some general probabilistic mechanisms behind missing values, one can also look at the shape of missing values in the data table. In the literature [16], three definitions of missing value patterns exist; namely *Univariate*, *Monotone* and *Arbitrary* pattern.

A *univariate* pattern (Figure 3.1a) of missing values means that one or several attributes (columns) contain missing values in exactly the same records and no other values are missing. When attributes can be organized in several groups $G_1, ..., G_k$, such that each group forms a univariate pattern and records with missing values in G_i have also missing values in G_{i-1}, for $i = k, ..., 2$, then we have a *monotone* pattern (Figure 3.1b). An *arbitrary* pattern, is anything else. Obviously, in order to visualize patterns of missing values, one has to permute columns and rows of the data matrix to create "rectangular regions of missingness".

It is very important to understand the patterns of missing values in a data set, because they might provide insight into why values are missing and relations of attributes that are missing in groups. As an example: A camera system fails to recognize the bar-code of a certain product, which in turn makes it impossible for the next two sensors to save their measurements to the database, resulting in two missing values. Additionally, identifying important patterns of missing values in the data can lead to using a different strategy for handling these missing values. However, in reality there are many more patterns that are now falling

under the category *arbitrary pattern*, but that are not arbitrary at all. Consider a data set with a Monotone pattern of missing values, now remove one value from a column that does not contain any missing values yet. The data set with the extra removed value falls under the arbitrary category, while in reality the data set is almost completely falling into the category of a monotone pattern. Another example, imagine that a survey takes place led by two volunteers, both volunteers ask the same ten questions to a hundred different people, but volunteer a asks the questions in order, and volunteer b asks the questions in reverse order. Due to time limitations, people start to drop out after the sixth question. The combined data set seems to have an arbitrary pattern of missing values, while if we look more closely we can identify two partitions of the data with both one monotone pattern of missing values.

To fill the gap between the definitions of missing value patterns, we introduce the concept of *Mixtures of Monotone patterns*. This requires a more precise definition of the *Univariate* and *Monotone* patterns.

Let us consider a data set D of size $N \times k$, with N the number of records in D and k the number of attributes in D, and a missing indicator matrix I. Here, $I_{ij} = 1$ if D_{ij} is missing, and 0 otherwise.

Definition 3.2 (Univariate pattern) *Missing values in D form a* univariate pattern *if and only if there exists a set of attributes A such that:*

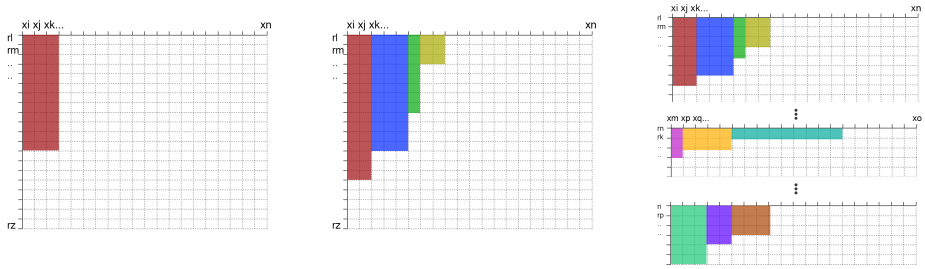
$$\forall x \in D : \{a : x_a \text{ is missing}\} = A \text{ or } \emptyset$$

So for all records in D, the record has either missing values in all attributes in the attribute set A or the record has no missing values.

Definition 3.3 (Monotone pattern) *A data set D has missing values in a* monotone pattern *if and only if there exists an ordering of all attributes A, $a_1 \dots a_k$ such that:*

$$\forall i \in \{1, ..., N\}, \forall j \in \{1, ..., k\} : I_{i,j} = 0 \Rightarrow I_{i,j+1} = 0, \dots, I_{i,k} = 0$$

Note that Definition 3.3 is a generalization of Definition 3.2, in other words, a



(a) Missing values in a univariate pattern.

(b) Missing values in a monotone pattern.

(c) Missing values in a k-monotone mixture pattern.

Figure 3.1: Data set of records (y-axis) and attributes (x-axis) with missing values denoted by the colored bars.

univariate pattern is a special case of a monotone pattern. A monotone pattern can also be seen as a collection of record groups, where each group of records has a univariate pattern of missing values. For example, a data set has twenty attributes and forty records, five of the records have attribute one and five missing this is denoted as: $(1, 5)$. Ten records have attributes one, five and nineteen missing: $(1, 5, 19)$, and twenty records have only attribute five missing: (5). The remaining five records are complete. The complete data set has a monotone pattern of missing values, which can be denotes as set $p = \{(5), (1, 5), (1, 5, 19)\}$. Each element in p stands for a univariate pattern that holds within a subset of the complete data set.

This way we can further generalize into a *Mixture of Monotone patterns* (Figure 3.1c).

Definition 3.4 (k-Monotone Mixture Pattern) *A data set D has missing values in a* k-monotone mixture pattern *if and only if here is a partitioning of D, $S = S_0 \dots S_{k-1}$ of size k such that $S_0 \cup S_1 \cdots \cup S_{k-1} = D$ and $\forall S_i, S_j \in S, i \neq j$: $S_i \cap S_j = \emptyset$ and $\forall S_i \in S : S_i$ has values missing in a monotone pattern.*

A univariate pattern can be seen as a rectangular area of missingness, a monotone pattern can be seen as a stack of adjacent rectangular regions and a monotone mixture pattern can be seen as a union of disjoint monotone patterns.

Note that **any** data set has values missing in a *k-Monotone Mixture Pattern* where $k \leq N$. When there exists a 1–*Monotone Mixture Pattern*, the pattern is completely monotone, when there exists only a high k mixture pattern, the pattern is close to arbitrary. In this manner, a transition between completely monotone and arbitrary patterns can be identified.

3.2.3 Analyzing Missing Value Patterns

It is possible to analyze a data set and identify the existing monotone mixture patterns using our novel *MMP-Finder* (Algorithm 3.1). In this algorithm, first a dictionary with all existing monotone patterns of missing values is built and sorted by the number of rows per pattern. Then, mixtures of monotone patterns are constructed by adding the next monotone pattern to an already existing mixture, or by defining a new mixture. The MMP-Finder uses a greedy approach to construct the mixtures of monotone patterns.

The complexity of the proposed greedy approach is $O(n + m^2)$, where n is the number of records and m is the number of unique sets of missing attributes. Of course $m \leq n$, since every record can have a unique set of attributes missing, but usually m is much smaller than n.

Using the MMP-Finder, all identified monotone patterns in the partitions of data set X are returned, together with the number of records and record indexes that belong to each monotone pattern. Notice that the solution returned is not a *unique solution*, it is possible that a specific univariate pattern can belong to multiple monotone patterns. For example, two monotone pattern sets are defined: $\{(1,5),(1,5,8)\}$ and $\{(4,5),(4,5,9)\}$, the next univariate pattern that occurs is (5), this pattern might belong to the first monotone pattern set, or the second. The proposed algorithm handles these choices in a greedy manner, the univariate patterns are handled in an order depending on the coverage, the pattern that covers most records is handled first, the pattern that covers the least records is handled last. This way it is very likely to identify "the biggest" monotone pattern. Since the missingness mechanism is usually not known, it is impossible to find the "correct" monotone patterns.

Algorithm 3.1 MMP-Finder

Given: A training set X with input attributes $x_0, \ldots, x_n$ containing missing values, a target attribute y

{Create a dictionary CM with all unique missing attribute combinations}
$CM = unique($**foreach records** $x_i \in X : Attr\_missing(x_i))$
{For each combination store the records with that combination of missing attributes}
$RecordsPerCombination = X[Attr\_missing(CM)]$
{Sort the combinations by size}
$sortedComb = sort(CM)$
$Mixtures = []; MixtureRecords = []$
{Construct the mixtures}
for all $comb \in sortedComb$ **do**
  for all $M \in Mixtures$ **do**
    {If the comb. is a sub or super set for all combinations in M add it to M}
    if $\forall c \in M : comb \subseteq c \vee comb \supseteq c$ **then**
      $Mixtures[M].append(comb)$
      $MixtureRecords_M.append(RecordsPerCombination[comb])$
      $Added = True$
      break
    end if
  end for
  if $\neg Added$ **then**
    {Add a new Mixture}
    $Mixtures.append([comb])$
    $MixtureRecords.append([RecordsPerCombination[comb]])$
  end if
end for
return $Mixtures$

Table 3.1: Textual summaries for data sets with Missing Values

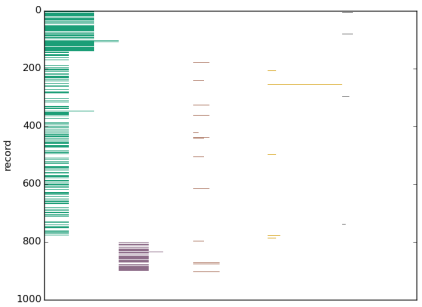
Data set	#Mixtures	Ratio of each mixture	Miss.%	Category
Post-oper.	1	[1.0]	0.033	Monotone
Wisconsin	1	[1.0]	0.023	Monotone
Dermato.	1	[1.0]	0.022	Monotone
Cleveland	2	[0.667 0.33]	0.020	Two monotone patterns
Adult	2	[0.776 0.224]	0.074	Two monotone patterns
Census	7	[0.948 0.030 0.006 0.001 ...]	0.527	Mostly monotone
Automobile	4	[0.826 0.087 0.043 0.043]	0.224	Mostly monotone
Hepatitis	7	[0.707 0.093 0.093 0.053 ...]	0.484	70% mono., 30% rand.
Mammogr.	5	[0.550 0.244 0.160 0.038 ...]	0.136	Monotone mixture
Bands	18	[0.39 0.259 0.086 0.086 ...]	0.323	60% two patterns
Wiki	116	[0.503 0.091 0.030 0.016 ...]	0.807	50% mono., 50% rand.
Marketing	39	[0.353 0.128 0.112 0.111 ...]	0.235	Random
Horse-colic	82	[0.221 0.061 0.058 0.044 ...]	0.981	Random

Once the monotone patterns and their support are known, it is easier to verify why certain attributes contain missing values, and whether there are relations between the various attributes inside the monotone patterns. This can not only provide valuable insight, but also help in choosing a good imputation or modeling algorithm.

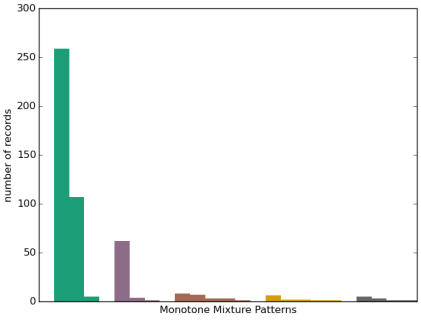
3.2.4 Analysis of Existing Data Sets

Fourteen data sets with missing values from the *UCI machine learning repository* [25] were analyzed using Algorithm 3.1. The output of the algorithm is shown in Table 3.1, and a visualization of the result is provided in Figure 3.2 and Figure 3.3. The visualization and textual summaries are generated directly from the output of the MMP-finder algorithm.

In Figure 3.2, the monotone mixture patterns found in the *Wiki* data set can be observed using two kinds of visualization techniques. In Figure 3.2a each record in the data set that contains missing values is labeled with a color and a position on the x axis. This way it is easy to observe where several monotone mixture patterns are located in the data set and if there are specific regions in the data set where these patterns occur. Additionally, the horizontal length of each bar depends on the number of attributes that are missing. For each mixture, the longest



(a) Visualization of missing values per record. Each column (color) represents a mixture of monotone patterns, the length of each bar is proportionate to the number of missing attributes versus the maximum number of missing attributes in its mixture.

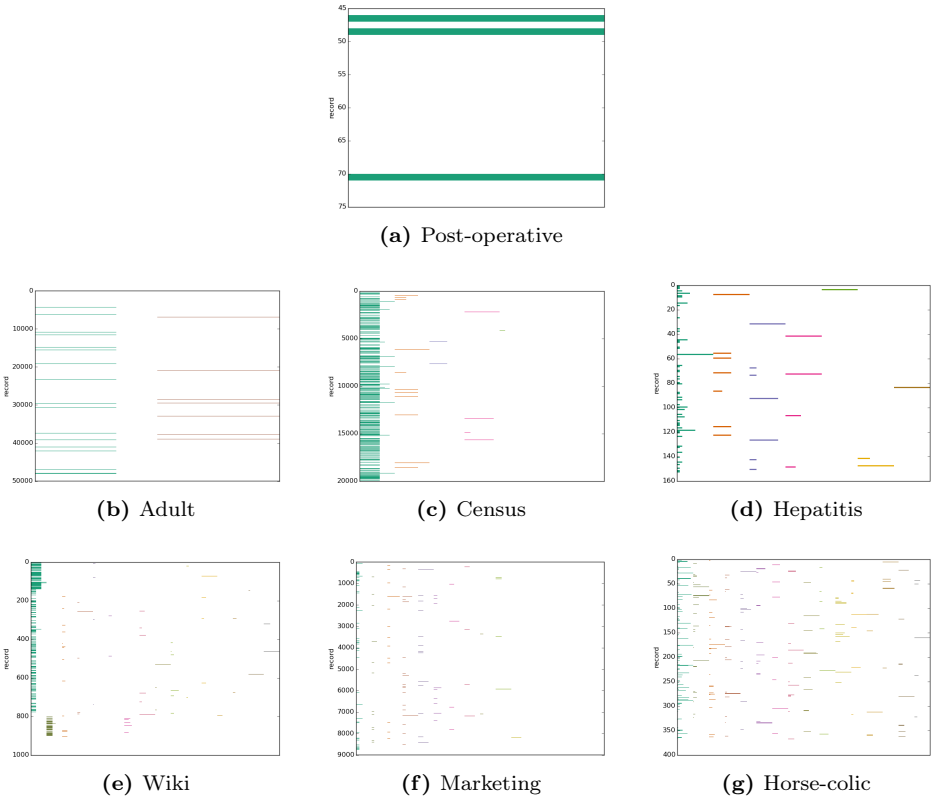


(b) Visualization by the number of records affected per pattern. Each color is a mixture of monotone patterns, each bar is a monotone pattern.

Figure 3.2: First five Monotone Mixture Patterns for the *Wiki* data set.

pattern (pattern with most attributes missing) has a length of one, and each other univariate pattern belonging to the same monotone pattern, has a length proportional to the ratio of missing attributes over the longest pattern.

This visualization technique, presenting the various monotone patterns in a data set, can be useful in understanding the underlying missing data mechanisms. For example, in the Wiki data set, the two monotone patterns that cover most of the records are located in a very specific order in the data set, which might be relevant information regarding the missing data mechanism. Even more specific, the three most occurring *univariate patterns* occur exactly after each other in the data set. In Figure 3.2b, the same patterns can be observed, but now in a histogram plot. The distribution of records belonging to each univariate pattern can be observed and the largest monotone patterns in terms of the number of records and in terms of the number of univariate patterns can be identified easily. Using this visualization technique it is easy to observe the different distributions in between the patterns. In Figure 3.3, a visualization of all the data sets with natural missing values is shown using the first visualization technique.



(a) Post-operative

(b) Adult **(c)** Census **(d)** Hepatitis

(e) Wiki **(f)** Marketing **(g)** Horse-colic

Figure 3.3: Visualization of data sets with natural occurring missing values.

3.3 Incremental Attribute Regression Imputation

Now that the patterns of missing values can be analyzed, the next step is repairing these missing values. In this section the proposed IARI algorithm is explained in detail and experimental results are discussed.

There are two ideas behind our method for incremental repair of training sets. First, attributes with missing values are repaired one by one, according to the priority of the attribute. The attribute with the highest priority is repaired first, the attribute with the lowest priority is repaired last. Second, the data used for repairing an attribute include all attributes that are already repaired and addi-

tionally the target attribute of the original data set. The choice of the repair algorithm is arbitrary, in principle any regression algorithm can be used here. In our experiments we used Random Forest [24], due to its superior accuracy, speed and robustness. Random Forest requires little to no tuning, which is very important when numerous models have to be developed without human assistance. Additionally, the Random Forest algorithm provides a heuristic for ranking attributes according to their importance. The IARI algorithm uses this heuristic for ordering the attributes.

It might seem counter-intuitive to include the target attribute in the set of predictors to impute an input attribute–it resembles a circular process. However, our goal is to repair a training set with help of any data we have. When the training set is fixed, a final model is trained and it can be applied to fresh data that were not used in the training process, so there is no circularity here. Moreover, results of our experiments demonstrate that including the target variable in the imputation process substantially increases the accuracy of the final model which is validated on data that were not used in the imputation process.

The IARI algorithm consists of two steps: initialization and main loop. During the initialization all attributes are split into two groups: those that contain no missing values (REPAIRED), and all others (TO_BE_REPAIRED). Assumed here is that the target attribute, y, contains no missing values so it falls into the REPAIRED group. Additionally, the set of attributes with missing values is ordered according to their importance. This is achieved in three steps. First, the training set is repaired with help of a simple imputation method which replaces missing values of continuous attributes by their mean values and missing values of discrete variables are replaced by their most frequent values. Second, a Random Forest model is built on the repaired training set to predict values of y. Finally, the model is applied to randomized out-of-bag samples to measure the importance of all attributes, as described in [18].

When the initialization step is finished, the algorithm enters the main loop which repairs attributes with missing values, one by one, in the order of their importance (from most to least important). To repair an attribute x, IARI creates a temporary training set which contains all attributes that are already repaired (including y) as predictors and x as the target. All records where the value of x is missing are

removed from this training set and, depending on the type of x, a classification or regression variant of the Random Forest algorithm is used to model x. Finally, the model is used to impute all missing values of x and x is moved from the TO_BE_REPAIRED to the REPAIRED set.

The pseudo-code of a generic version of the *IARI* algorithm is provided in Algorithm 3.2.

Algorithm 3.2 Incremental Attribute Regression Imputation

Given: A training set X with input attributes $x_1, \ldots, x_n$, a target attribute y, and a classification or regression algorithm ALG

```
Initialization:
for all attributes x_i ∈ X do
  Nmissing[i] = Count_missing(x_i)
  Importance[i] = ImportanceMeasure(X, x_i, y)
end for
REPAIRED = y ∪ {All attributes x_i where Nmissing[i] = 0}
TO_BE_REPAIRED = {All attributes x_i where Nmissing[i] > 0}
while TO_BE_REPAIRED != ∅ do
  Repair_Attribute = SELECT_X_i(TO_BE_REPAIRED, Importance)
  Repair_Target = Delete_Missing_Values(Repair_Attribute)
  Model = ALG.train(REPAIRED, Repair_Target)
  for all records A_j ∈ Repair_Attribute do
    if is_missing(A_j) then
      A_j = ALG.predict(REPAIRED[j])
    end if
  end for
  REPAIRED = REPAIRED ∪ Repair_Attribute
  TO_BE_REPAIRED = TO_BE_REPAIRED \ Repair_Attribute
end while
return REPAIRED
```

3.3.1 Existing Imputation Algorithms

There are many ways of dealing with missing data when building a regression or classification model. Some of the most popular methods are:

***Complete Case Analysis* (CCA):** This method simply ignores all records that

have missing values and selects only records with no missing values [27, 21]. When the percentage of complete records is relatively high and the data are missing at random or completely at random, this method does not affect model accuracy. However, if the amount of missing data is large, the prediction accuracy will be low (not enough complete cases), and when the data are missing not at random, then this method generates bias.

***Missing Indicator Variable* (MIV):** This method uses a dummy variable as an indicator for missing values [27]. For every variable that might be missing, a dummy variable is introduced, where the value of this dummy variable is 1 when the input variable is missing and 0 when the input variable is not missing. While this method is more efficient than the Complete Case Analysis, it can also create bias in the final model.

***Predictive Value Imputation* (PVI):** PVI replaces missing values by some estimates of their values [28]. In many cases the unconditional mean is used (the mean value of all non-missing values of the attribute) or a conditional mean (the mean of a specific group of records where the record with a missing value belongs to). The problem with this method is that the predictive values are always derived from the complete cases and that might introduce some bias. However, some additional mechanisms can be added to PVI which lower this bias. For example, PVI might use the conditional mean over the k-nearest neighbors of a record with a missing value, and then the bias can be limited by first imputing the data set with unconditional mean and then using the k-nearest neighbors on the completed data set to predict the values of the originally missing data. By counting the number of missing data in the neighbors, one can create a weighted average that incorporates the uncertainty of the measurements. There are several other methods to do single-value predictive imputation like *hot-deck imputation*, *cold-deck imputation* and *last observation carried forward*, where the data set is sorted on specific variables and when a missing value is encountered, the value is replaced by the value of its predecessor.

***Regression Imputation* (RI):** Regression Imputation [28] is a PVI variant where regression models (Support Vector Machines, Random Forests, etc.) are used to estimate the imputed value. One way is to build the models to estimate

the missing values using the complete cases. However, it is usually better to also incorporate the non-complete cases by first imputing the missing values with a more simple imputation method (like the unconditional mean). In the first case (using only complete cases), there might be too few complete cases to generate good models, in the latter case there is a danger of bias by training the model with imputed (wrong) data.

***Multiple Imputation* (MI):** This is a general imputation framework by Rubin et al. [20, 15, 22, 29]. The idea is to generate multiple versions of imputed (completed) data sets, which result in multiple models. Each model is then combined into a final predictor. The framework uses a single value imputation algorithm of choice and a random component that represents the uncertainty of the imputation. By creating multiple imputed data sets, the distribution of the imputed values will reflect the distribution of the already known values and therefore reduce bias. This method allows any non-deterministic imputation algorithm to be used. After imputing the data set several times, creating several copies, a model is being built for each complete data set. The results of each model are combined using Rubin's Rules [20]. The combined result leads to less biased and more accurate predictions. One of the major advantages of MI is that it can be used with almost any imputation algorithm. Because of this, MI is not added in the comparison because each of the imputation algorithms can be wrapped with Multiple Imputation.

Most of the above methods can also be used for handling missing data at prediction time. The CCA method is here an obvious exception, but imputation or using dummy variables are valid ways to deal with missing values at prediction time. It should also be mentioned that in addition to the classical "off-line" scenario, where the training set is fixed and is not changing over time, some researchers were considering an "on-line" scenario, where the model is continuously updated while processing a stream of data [30].

In this chapter a novel strategy is proposed that uses regression models in an attribute wise algorithm to impute missing values in the training stage using the target attribute as one of the predictors. The proposed algorithm is compared

with commonly used imputation methods and an imputation method that also uses regression models: *Regression Imputation*.

3.3.2 Experimental Setup

To compare the existing algorithms with our approach we used five, very different, data sets from various Machine Learning Repositories: *Digits*, *Cover Type*, *House 16H*, *Page Blocks*, and *Concrete Compressive Strength*. For a complete overview of these data sets, see the public IARI repository, [31].

Parameters

In our experiments, a popular implementation of the Random Forest algorithm, that comes with the *Scikit-learn* Python package, [32], is used. The key learning parameter, the number of estimators, was set to 100, and the remaining parameters had default values.

For each data set several experiments are done with 75% of the attributes containing missing values and 25% of the attributes (randomly chosen) containing no missing values. The amount of missing values in the attributes with missing data, was set to $10, 20, 30, 40, 50, 60$ percent and for each setup we run 20 experiments using different random seeds. In each experiment, the complete data set was split in a training (80%) and a test set (20%). The deletion of values, repairing the training set and final modeling was performed on the training set. The test set was used to estimate the accuracy of the final model. When removing values from the training set we used two strategies: "missing at random", *MAR*, where values were removed uniformly at random, and "missing not at random", *MNAR*, where only values bigger than the median value of the attribute, were removed uniformly at random.

Performance Indicators

Two aspects are measured for the quality of the imputation. First, the accuracy of the final model, that was trained on the repaired data set, is estimated with help

of cross-validation. The accuracy was measured either by the ratio of correctly classified cases (in case of classification) or by the *coefficient of determination*, R^2, (in case of regression):

$$R^2 = 1 - \frac{\sum_i (p_i - y_i)^2}{\sum_i (y_i - \bar{y})^2}$$

where y_i denotes the target value and p_i the predicted value.

This score indicates how well the model fits the test data. The maximal value of R^2 is 1, meaning the perfect fit; values smaller than 1 reflect the error. Furthermore, the R^2 and accuracy scores of each data set are measured on final models that were developed with three algorithms: Random Forests, Support Vector Machines and Gradient Boosted Decision Trees. This is to demonstrate that the value of R^2 (or the accuracy score) depends on the regressor or classifier that is being used in the final modeling, and that it not always reflects the quality of the imputation itself.

Secondly, we measured the quality of the approximation of the imputed values. As all the imputed variables were numeric, we used the *Root Mean Squared Error*, *RMSE*, to measure the difference between the observed and imputed values:

$$\mathrm{RMSE} = \sqrt{\frac{\sum \left(v_{\mathrm{observed}} - v_{\mathrm{imputed}}\right)^2}{n}}$$

To make the comparison of results over various data sets meaningful, the attributes of all training sets are standardized by centering them around 0 and dividing by their standard deviations. As the last indicator of an algorithm's performance, the execution time is measured. For bigger data sets the CPU time might be an issue to consider.

3.3.3 Results

For each data set, we performed 12 experiments: one for each of the percentage levels of missing values (from 10 to 60) combined with the type of missingness

(MAR or MNAR). Each experiment was repeated 20 times (with different random seeds) and the results were averaged. Additionally, for each reconstructed training set, we run three algorithms, Random Forests, Support Vector Machines and Gradient Boosted Decision Trees, to build the final models.

The results of our experiments, the accuracy of the final model (R^2 or the ratio of correctly classified cases) and the accuracy of imputation ($RMSE$), are presented in the following subsection. Each row contains averaged results of 20 runs of the same experiment with different random seeds. The amount of missing values and the type of missing values (MAR or MNAR) are shown as well. For the MNAR model we used the missing percentages 20%, 40%, and 60% as upper bounds for the percentage of missing values per attribute, but it was not always possible to delete that many values of the attribute due to the restriction of deleting only values bigger than the median. Let us note, that it may happen that the fraction of records with a value of an attribute bigger than its median might be arbitrarily small, e.g., when an attribute is almost constant. Moreover, in the results presented below, we show the average number of missing values taken over all attributes with missing values.

For the first three data sets (Cover Type, Digits and Houses 16H) we show the results from the Random Forest final model; for the remaining data sets and final model options please see Appendix A.

Each table contains several columns. The first two columns contain information about the percentage of missing values and the type of missingness. The third column, *Ref.*, contains the accuracy of the model trained on the original complete data set: either R^2 for regression problems or classification accuracy for classification problems. The following columns contain results of various imputation methods: Imputation by Mean, Imputation by Median, Imputation by Most Frequent, Predictive Value Imputation using 2-nearest neighbor over a data set imputed by the Mean, Regression Imputation using Random Forests and last but not least, our own algorithm: IARI. Entries in boldface are significantly better than all other entries with the same settings. The significance is tested using the *t-test*, with significance level $p = 0.05$. The absence of a bold entry in the row means that none of the results were significantly better than the others.

Cover Type data set Results

In Table 3.2 and 3.3 the accuracy of the model (Accuracy Score) and the quality of imputation ($RMSE$) are shown for the imputation algorithms on 40.000 instances of the Cover Type data set.

Table 3.2: Model Accuracy Score on the Cover Type data set with 40.000 instances using Random Forests

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
4	MNAR	0.911	0.887	0.879	0.876	0.886	0.886	**0.893**
6	MNAR	0.911	0.871	0.864	0.860	0.868	0.868	**0.881**
10	MNAR	0.911	0.815	0.809	0.803	0.806	0.805	**0.839**
12	MNAR	0.911	0.670	0.678	0.656	0.657	0.663	**0.693**
10	MAR	0.911	0.893	0.899	0.900	0.900	0.896	**0.907**
20	MAR	0.911	0.874	0.887	0.886	0.883	0.880	**0.899**
40	MAR	0.911	0.834	0.859	0.858	0.845	0.845	**0.878**
60	MAR	0.911	0.776	0.824	0.822	0.787	0.799	**0.847**

Table 3.3: Imputation Quality (RMSE) of each Imputation Algorithm on the Cover Type data set with 40.000 instances

Miss.%	Type	Mean	Median	Freq.	PVI NN	RI	IARI
4	MNAR	0.755	0.762	0.774	0.755	0.745	**0.721**
6	MNAR	0.786	0.795	0.813	0.786	0.776	**0.760**
10	MNAR	0.848	0.852	0.867	0.847	0.838	**0.791**
12	MNAR	0.894	0.884	0.889	0.894	0.894	**0.877**
10	MAR	0.271	0.277	0.294	0.261	0.225	**0.176**
20	MAR	0.380	0.389	0.414	0.370	0.330	**0.266**
40	MAR	0.540	0.552	0.588	0.533	0.496	**0.422**
60	MAR	0.661	0.676	0.718	0.658	0.630	**0.564**

Table 3.4: Execution time of Imputation Algorithms on the Cover Type data set with values 50% MAR in seconds.

Mean	Median	Freq.	PVI NN	RI	IARI
0.03	0.11	0.48	61.47	381.75	119.12

From our test results we can observe that the maximum average number of MNAR values we can delete from each attribute is around the 12%. Which implies that approximately 88% of the data set is filled with values below or equal the median of each attribute (probably 0). In Table 3.4 the execution time for each algorithm is shown for the case of 50% values MAR, which is representative for all the tests on this data set. Our approach is not the fastest, Replace by Median, Replace by Mean and Replace by Most Frequent are almost instant while PVI, RI and IARI are more complex and take some time. The execution time is mostly dependent on the size of the data set and on the number of attributes, and not so much on the number of missing values.

Digits data set Results

In Table 3.5 the *accuracy* of the models created using the different imputed data sets as training set is shown.

Table 3.5: Model Accuracy Score on the Digits data set using Random Forests

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
16	MNAR	0.972	**0.969**	0.967	0.953	0.967	0.967	0.968
25	MNAR	0.972	**0.966**	0.951	0.904	0.954	0.961	0.947
27	MNAR	0.972	**0.949**	0.923	0.815	0.934	0.932	0.944
20	MAR	0.972	0.964	0.963	0.961	0.967	0.968	**0.970**
40	MAR	0.972	0.954	0.957	0.952	0.959	0.960	**0.962**
60	MAR	0.972	0.944	0.943	0.934	0.944	0.948	**0.953**

Table 3.6: Imputation Quality (RMSE) of each Imputation Algorithm on the Digits data set

Miss.%	Type	Mean	Median	Freq.	PVI NN	RI	IARI
16	MNAR	0.608	0.649	0.752	0.566	0.565	**0.479**
25	MNAR	0.858	0.903	1.037	0.841	0.829	**0.646**
27	MNAR	0.974	0.994	1.103	0.960	0.963	**0.850**
20	MAR	0.380	0.399	0.511	0.299	0.316	**0.231**
40	MAR	0.537	0.564	0.721	0.470	0.472	**0.345**
60	MAR	0.658	0.692	0.883	0.619	0.608	**0.451**

In Table 3.6 the $RMSE$ values for every imputation algorithm are presented for all the combinations of missing data percentage and missing data type. Our IARI approach outperforms the other imputation algorithms in most of the MAR cases with respect to the accuracy. In the MNAR cases our algorithm works well but imputation by Mean sometimes has a slightly better accuracy for the Random Forest model.

Houses 16H data set Results

In Table 3.7 the R^2 scores are shown and in Table 3.8 the $RMSE$ results are shown for the imputation algorithms on the Houses 16H data set.

Table 3.7: Model Accuracy Score (R^2) on the Houses data set using Random Forests

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
20	MNAR	0.636	0.604	0.598	0.580	0.606	0.603	**0.617**
40	MNAR	0.636	0.534	0.491	0.485	0.511	0.520	0.531
49	MNAR	0.636	−0.277	−0.287	−0.545	−0.405	**−0.171**	−0.450
20	MAR	0.636	0.604	0.599	0.586	0.610	0.598	**0.620**
40	MAR	0.636	0.544	0.533	0.511	0.552	0.521	**0.590**
60	MAR	0.636	0.423	0.402	0.375	0.458	0.414	**0.536**

Table 3.8: Imputation Quality (RMSE) of each Imputation Algorithm on the Houses data set

Miss.%	Type	Mean	Median	Freq.	PVI NN	RI	IARI
20	MNAR	0.486	0.517	0.709	0.485	0.428	**0.342**
40	MNAR	0.785	0.801	1.007	0.787	0.753	**0.587**
49	MNAR	0.956	0.925	1.134	0.955	0.954	**0.927**
20	MAR	0.386	0.395	0.542	0.370	0.328	**0.280**
40	MAR	0.545	0.558	0.764	0.535	0.487	**0.412**
60	MAR	0.669	0.685	0.925	0.665	0.625	**0.531**

3.4 Attribute Selection and Sorting Methods

For the IARI algorithm the most important attributes are selected to be imputed first by using the out-of-bag samples provided by the random forest algorithm. However, the question is if this is the best possible order of imputation. Comparing the scores of IARI using this attribute ordering with the scores of IARI using the exact opposite attribute ordering (least important attribute first) shows that the results are only slightly worse for the latter case. This implies that the order of attribute repair does matter, but is not a key factor in the algorithm.

3.4.1 Greedy Model Accuracy Selection

Another sorting or attribute selection method would be to repair each attribute first, using the already repaired attributes, and select the attribute that improves the accuracy of the model the most. This selected attribute is then added to the training set and the procedure is repeated till all remaining attributes do not improve the accuracy any further. This greedy approach leads to an algorithm where one can decide to stop imputing the attributes once the model does not improve any longer. Effectively, giving an algorithm that not only repairs the attributes of the data set but also selecting which attributes are useful to impute and which attributes are not improving our model and so might be better left out.

Running this greedy version of our algorithm on the Digits data set we get the R^2 results of Table 3.9.

The greedy version of IARI uses on average only 22 of the 64 attributes that are available. It is interesting to see that the results of our greedy approach are actually worse than the results of our non-greedy approach for the Digits data set. The reason, is most likely that too many attributes are ignored (discarded) by the greedy approach. It can be the case that adding an attribute to our model might not improve the model's accuracy immediately, but adding this attribute in combination with another attribute might still improve the model. This can clearly be seen in the following example where a model is trained on the function

Table 3.9: Model Accuracy Score (R^2) for the greedy IARI approach on the Digits data set

Miss.%	Type	Ref.	IARI	GREEDY
8	MNAR	0.964	**0.961**	0.954
16	MNAR	0.964	**0.953**	0.949
23	MNAR	0.964	**0.950**	0.912
25	MNAR	0.964	**0.926**	0.907
27	MNAR	0.964	**0.915**	0.908
10	MAR	0.964	0.956	0.952
20	MAR	0.964	0.951	0.951
30	MAR	0.964	**0.955**	0.943
40	MAR	0.964	**0.952**	0.941
50	MAR	0.964	**0.948**	0.935
60	MAR	0.964	**0.934**	0.921

Table 3.10: Model Accuracy Score (R^2) for the greedy IARI approach on the Houses data set

Miss.%	Type	Ref.	IARI	GREEDY
8	MNAR	0.636	0.630	**0.632**
8	MNAR	0.636	0.617	**0.624**
8	MNAR	0.636	0.585	**0.605**
8	MNAR	0.636	0.531	**0.574**
8	MNAR	0.636	−0.450	**0.016**
10	MAR	0.636	0.627	**0.635**
20	MAR	0.636	0.620	**0.631**
30	MAR	0.636	0.608	**0.624**
40	MAR	0.636	0.590	**0.614**
50	MAR	0.636	0.567	**0.599**
60	MAR	0.636	0.536	**0.580**

$y = OR(x_1, XOR(x_2, x_3))$. If the model is trained using only x_1, the final accuracy will be 75%. When adding x_2 to the training set (after fixing possible missing values) the accuracy of the model will not improve. However, by also adding x_3 to the training set the model's accuracy will increase to 100% (under the assumption that there are sufficient samples to learn from).

This simple example tells us that combinations of attributes might be valuable

even if all the single attributes are not giving any improvement on the final model. In the case of the Digits data set, this might be the case since each attribute only stands for a single pixel, and combinations of pixels create the information needed to see what digit the image represents, while single pixels might not give any information at all.

The greedy algorithm is also evaluated on the Houses 16H data set, which seems a more suitable data set for such an approach. The data set has 16 attributes and the greedy algorithm uses on average 13 of them. The R^2 scores of the original IARI and the greedy modification can be seen in Table 3.10. In this case the greedy algorithm performs slightly better than the original IARI algorithm. It clearly depends on the data set and the dependencies between the attributes if the greedy attribute selection works or not.

3.4.2 Greedy Imputation Quality Selection

Another possibility to optimize the IARI algorithm is to select the features to be repaired on the *repairability* of each attribute. With repairability we mean how well a specific attribute can be repaired. This can be measured by the out-of-bag error provided by the Random Forest models used to repair each attribute with. The main idea behind this approach is that if the attributes are repaired in an order of attributes that can be repaired as good as possible first, it might be possible to repair the remaining attributes even more accurately and as such, the final model will have a lower $RMSE$. In Figure 3.4 the $RMSE$ of the IARI algorithm and the repairability selection variant of IARI (named REPAIR) are presented. The REPAIR algorithm has a slight improvement in the RMSE score, but the modified algorithm also takes more time due to the $n * (n - 1)$ models it trains.

3.5 Conclusions

A new concept to analyze and visualize missing value patterns in data sets is proposed and it is shown that with a greedy method called MMP-finder, a data

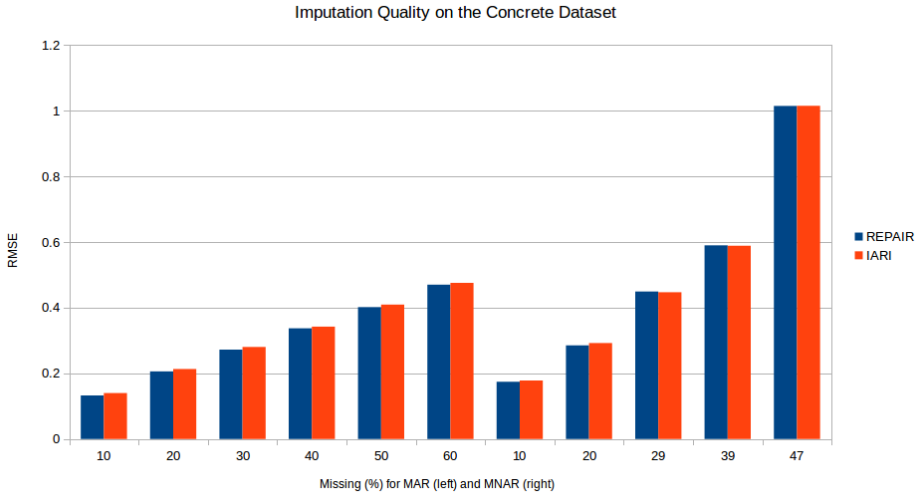


Figure 3.4: Imputation Quality ($RMSE$) on the Concrete data set

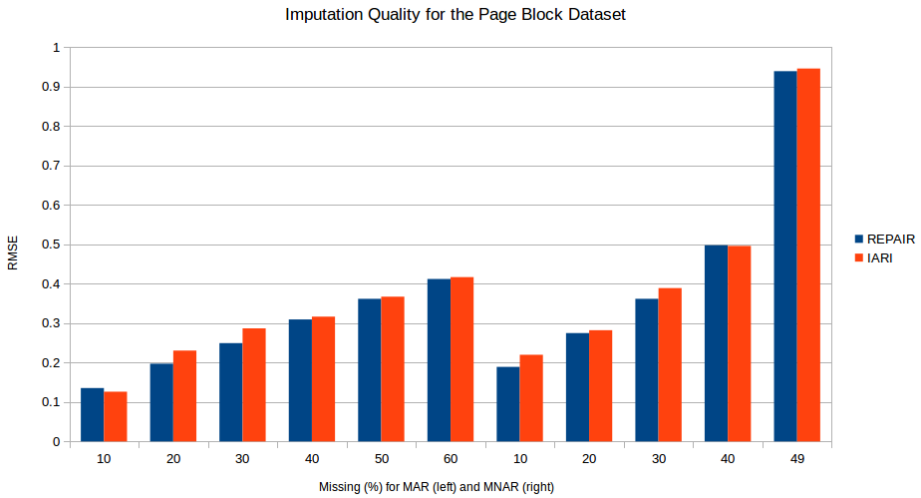


Figure 3.5: Imputation Quality ($RMSE$) on the Page data set

set with missing values can be analyzed. Using the concept of *k-Monotone Mixture Patterns*, a better in-depth understanding of the underlying patterns of missing values or unique values can be obtained.

Furthermore, a missing value generator is developed and made publicly available to make it possible to test and compare different algorithms on a wide set of both generated and natural data sets. The proposed missing value generator can be used as a tool to generate benchmark data sets for algorithms that handle certain kinds of missing values, giving insight in what kind of situations an algorithm might work well and in what situations certain algorithms would not perform well. For future work, it would be interesting to incorporate various mechanisms of missing values into the generator to also cover *MNAR* situations. Another interesting extension to the proposed analysis algorithm would be to include expert knowledge and assumptions to come up with the most likely partition of monotone patterns.

In addition, a novel algorithm, IARI, is proposed for imputing missing values into training sets. IARI can handle both regression and classification problems. The key advantage of IARI over other imputation methods is the superior accuracy of the final models which are trained on the repaired training sets, and more accurate reconstruction of missing values. On the other hand, IARI requires much more computing resources than its alternatives: 2-3 orders of magnitude. Fortunately, the main algorithm behind IARI, Random Forest, can be efficiently distributed along multiple nodes, significantly reducing the real (wall clock) computation time.

In principle, IARI is a generic algorithm which can be configured in various ways by changing the measure of importance of attributes, ordering of attributes, and the base algorithm used for imputation. Also the initialization step, where only attributes with no missing values are used as a starting set of predictors, can be modified: sometimes adding to this set several attributes with just a few missing values and removing incomplete records from it, lead to better results.

During our experiments with IARI, it was noticed that sometimes a simple imputation method may lead to better results than those obtained by IARI. This

happens in case of the *Digits* data set, where values were removed "not at random", see the *IARI repository* [31] for additional details. As expected, the quality of IARI approximations of missing values was always significantly better than those obtained by imputing means, but surprisingly, the opposite holds for the quality of the corresponding final models. This could be caused by the nature of the classification problem and the fact that Random Forest is not suitable for image classification. Almost in all other cases, the IARI algorithm outperforms other imputation methods: both in terms of the accuracy of imputation and the accuracy of the final model. In most real world cases it is difficult to determine how well a certain imputation algorithm will work. The quality of imputation depends a lot on the data set and the reason of why values are missing. However, when little is known about a data set, the IARI algorithm is a good choice to start with.

Both the analysis and visualization methods and the IARI algorithm are publicly available on Github in the *IARI repository* [31] and *MisVis repository* [33].

4

Outlier Detection in High-Dimensional Big Data

Using the preprocessed input data, a lot of data science can already be conducted to provide valuable insights for the domain experts. Unsupervised learning techniques such as clustering and outlier detection can warn the operator for incoming issues or allow the operator to make educated guesses about the process parameters to adjust.

In this chapter we focus on outlier detection in high-dimensional data, which is important in the complex manufacturing processes of BMW, since they are interested in knowing when and where incoming steel coils show anomalous measurements. This knowledge enables BMW to decide which coils to process, which to process with careful settings and which to reject completely. Outlier detection in high-dimensional data is a challenging yet important task, as it has applications in, e.g., fraud detection and quality control. In these highly complex and high-dimensional data sets, existing methods are likely to overlook important outliers because they do not explicitly take into account that the data is often a mixture distribution of multiple components. Proposed, in this chapter, is an algorithm specifically tailored to find outliers in high-dimensional data sets which is also used in the real-world application of BMW.

4. ANOMALY DETECTION

In this chapter the challenge of anomaly detection in high-dimensional datasets is introduced and an algorithm, GLOSS, that performs *local subspace outlier detection using global neighborhoods*, is proposed. Experiments on synthetic data demonstrate that GLOSS more accurately detects local outliers in mixed data than its competitors. Moreover, experiments on real-world data show that our approach identifies relevant outliers overlooked by existing methods, confirming that one should keep an eye on the global perspective even when doing local outlier detection.
The content of this chapter is primarily based on publication [34].

4.1 Introduction

Outlier detection [35] is an important task that has applications in many domains. In fraud detection, for example, a bank could be interested in detecting fraudulent transactions; in network intrusion detection, it could be of interest to automatically detect suspicious network events; in a manufacturing plant, identifying raw materials or products with strongly deviating properties could be useful as part of quality control. In each of these applications, the data is *high-dimensional* and each *data point* is a potential outlier.

Many techniques for outlier detection have been proposed and studied. Many traditional outlier detection methods [36] are parametric and thus make strong assumptions about the data. Moreover, data points are always considered as a whole and relative to all other data points, which strongly limits the accuracy of these methods on high-dimensional data. Outlier detection in complex, high-dimensional data is an inherently hard problem, as data points tend to have similar distances due to the infamous 'curse of dimensionality'. To address both this problem and the limitations of (global) outlier detection, *local* outlier detection methods [37, 38, 39] have been proposed over the past few decades. These methods are distance- or density-based, and assign outlier scores based on the distance of a data point to its closest neighbors relative to the local density of its neighborhood. To further improve on this, local subspace outlier detection methods [40, 41, 42] have been introduced. They search for local outliers within so-called *subspaces*, i.e., subsets of the complete set of features. This results in each outlier being reported together with a corresponding subspace in which it is far away from its neighbors. Existing local outlier detection approaches, however, are bound to overlook outliers when the data is *a mixture of high-dimensional data points drawn from different data distributions*. That is, as we will show, a local neighborhood found *within a given subspace* may very well include data points from different components of the mixture, which might result in clear outliers hiding in the crowd of a different component. This is especially relevant when the individual components of the mixture are unknown and hence the dataset has to be analyzed as a whole.

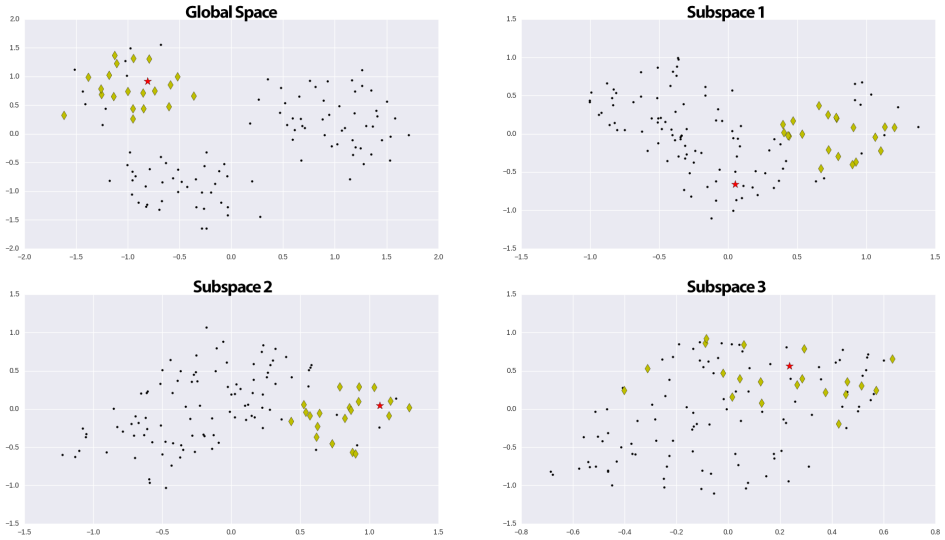


Figure 4.1: Dataset with six dimensions, consisting of a mixture of samples from three distributions. Shown are the global $6D$ space projected onto $2D$ (top left), and three orthogonal $2D$ subspaces (other). The implanted outlier (red star) can only be detected in Subspace 1 (top right) if local outlier detection uses its global neighbors (yellow diamonds) instead of its subspace neighbors (dots nearby).

We encountered this exact situation in an ongoing collaboration with the BMW Group, where our aim is to identify steel coils strongly deviating in terms of their material properties. The data is very high-dimensional, as it contains hundreds of measurements per coil, but is also known to be a mixture of samples from different distributions: the steel coils have different grades and come from different suppliers. Unfortunately, part of this information is not available in the data and we therefore had to analyze the complete, mixed data. However, what is a 'normal' measurement for one type of coil can be a clear deviation for another type of coil; therefore, existing outlier detection methods did not perform well. Section 4.4 will show examples of relevant outliers detected by our approach that were not found by existing methods.

Figure 4.1 illustrates the problem that we consider on a synthetic dataset. The data consists of three normally distributed clusters in six dimensions; the generative process (and experiments on generated data) will be described in detail in

Subsection 4.3.1. When considering all data points, the data point depicted by the red star is not a local outlier in any of the subspaces, neither in the global nor in any of the two-dimensional subspaces (only three shown). However, when only considering *the data point's neighbors in the global space*, here depicted with yellow diamonds, we can observe that the red star is a clear outlier in the 2D subspace shown in the top right plot. *it happens to be close to data points from other components, but is far away from data points from the component it belongs to and is therefore an outlier.* As we will show in Subsection 4.3.1, existing algorithms are unable to detect such outliers, especially in high-dimensional data, whereas our method can.

Approach and contributions Our first contribution is the formalization of the *Local Subspace Outlier in Global Neighborhood* problem. That is, we propose to combine local subspace outlier detection with neighborhoods selected in the global data space. The purpose of using global neighborhoods is to assess the degree of outlierness of a given data point relative to other data points *belonging to the same mixture component*, avoiding the possibility that outliers can hide among members of other components of the mixture distribution. Although global outlier detection in high-dimensional data is tough because of very similar distances, we argue and empirically show that globally selecting a neighborhood is nevertheless feasible. Following this, our second contribution is the introduction of the GLOSS algorithm, which combines our ideas on outlier detection using global neighborhoods with techniques from LoOP [39] and HiCS [40].

Given a dataset, it computes the probability that a data point is an outlier according to the problem definition. Moreover, it does so for all feature subspaces deemed relevant and hence also provides information about the subspace(s) in which a data point is considered to be an outlier. Although GLOSS is technically a seemingly straightforward extension of LoOP, the experiments will demonstrate this is conceptually an essential extension nevertheless.

Finally, the third contribution of this work is an extensive set of experiments on both synthetic and real-world data, in which we evaluate GLOSS and compare its performance to its state-of-the-art competitors. The experiments demonstrate that the use of global neighborhoods enable the discovery of outliers that would

otherwise be left undetected, without sacrificing detection accuracy on 'regular' outliers.

4.1.1 Related Work

Although most previous work on outlier detection has been done in statistics, there are also *clustering*-based [43], *nearest neighbor*-based [44], *classification*-based [45] and *spectral*-based [46] outlier detection algorithms. Statistical approaches can be categorized as: *distribution*-based [36], where a standard distribution is used to fit the data; *distance*-based [47], where the distance to neighboring points are used to classify outliers versus non-outliers; and *density*-based, where the density of a group of points is estimated to determine an outlier score. While classification, clustering- and distribution-based algorithms aim to find global outliers by comparing each data point to (a representation of) the complete dataset, distance- and density-based algorithms detect local outliers. We next describe the methods most relevant to our work:

Local Outlier Factor (LOF) [37] was the first algorithm to introduce the concept of *local density* to identify outliers. The authors also claim that they are the first to use a (continuous) 'outlier factor' rather than a Boolean outlier class.

Local Correlation Integral (LOCI) [38] detects outliers and groups of outliers (small clusters) using the *multi-granularity deviation factor* (MDEF). If a point differs more than three standard deviations from the local average MDEF, it is labeled as outlier.

Local Outlier Probabilities (LoOP) [39] is also similar to LOF but does not provide an outlier factor. Instead, it provides the probability of a point being an outlier using the *probabilistic set distance* of a point to its k-nearest neighbors. Given this distance and the distances of its neighbors, a *Probabilistic Local Outlier Factor* (PLOF) is computed and normalized. We will build upon LoOP in this work.

Subspace Outlier Detection (SOD) [41] is an algorithm that searches for outliers in meaningful subspaces of the data space or even in arbitrarily-oriented subspaces [42]. Other work in the area of spatial data uses special spatial attributes to define neighborhood and usually one other attribute to find outliers that deviate in this attribute given its spatial neighbors [48].[1]

Outlier Ranking (OutRank) [50] determines the degree of outlierness of points using subspace analysis. For the analysis of subspaces it uses clustering methods and subspace similarity measurements.

High Contrast Subspaces(HiCS) [40] is a state-of-the-art algorithm that searches for high contrast subspaces in which to perform local outlier detection. It uses LOF as the local outlier detection method for each such subspace, but other algorithms could also be used.

Other recent work such as [51] combines density clustering with local and global outlier detection. We will empirically compare to LOF, HiCS, and two variants of LoOP in Section 4.3, as these are well-studied and representative of the state-of-the-art in local (subspace) outlier detection.

4.2 Global Local Outliers in SubSpaces

4.2.1 Problem Definition

Many outlier detection (and data mining) algorithms assume—either implicitly or explicitly—that the data is an i.i.d. sample from some underlying distribution. That is, assumed is a dataset D_1 drawn from some fixed distribution q_1, denoted $D_1 \sim q_1$. Given this, global outliers can be found by approximating q_1 from the data, estimating $P(d \mid q_1)$ for all $d \in D_1$, and ranking all data points according to the resulting probabilities or scores.

In practice, however, many datasets are mixture distributions of multiple components. Consider for example a dataset D_2 consisting of a mixture of two components C_1 and C_2, drawn from two different distributions, i.e., $D_2 = C_1 \cup C_2$,

[1] More details and a comparison of these algorithms can be found in [49].

$C_1 \sim q_1$, and $C_2 \sim q_2$. Globally scoring and ranking outliers now becomes a very challenging task, as identifying the underlying distributions is a hard problem and different components may have different characteristics (such as overall density, attribute-value marginals, etc.).

Local outlier detection algorithms address this problem by considering distances or densities *locally* in the dataset, i.e., within the *neighborhood* of each individual data point. Although this approach generally works well, it has the disadvantage that it breaks down on high-dimensional datasets, for which all distances become similar; no data points are much further apart than others.

This problem can be addressed by using a local subspace outlier detection algorithm such as HiCS [40]. That is, given a dataset D consisting of data points over a feature space $\mathcal{F}$, these methods search for local outliers within feature subspaces $F \subset \mathcal{F}$. Each reported outlier is associated with a subspace F, explaining in which features the data point is different from its neighbors.

However, as argued in the Introduction, this approach suffers from a severe limitation: *existing approaches do not take into account that datasets may be mixtures of multiple components.* That is, when searching for local outliers within a feature space F, the density is locally estimated using a neighborhood determined *using the dataset projected onto the feature subspace only.* Unfortunately, as we will see next, this may have very undesirable side-effects.

That is, consider again our mixture dataset D_2. Suppose that a data point $o \in C_1$, i.e., drawn from q_1, is a clear outlier in a (small) subspace F, but its values for F are very normal for data points drawn from q_2. Then outlier o may go completely undetected by using existing algorithms:

1. First, because the data is high-dimensional, global outlier detection methods do not consider o to be far away from other data points in C_1 (o is only different in the feature set F);
2. Second, local outlier detection suffers from the same problem when considering all features;
3. Finally, local subspace outlier detection will not find the outlier either: the neighborhood of o based on D_2 projected onto F *consists of members of*

component C_2. Although o does not belong to that component, it is in fact very close to those 'neighbors' and is therefore not considered an outlier!

Summarizing, existing methods cannot detect outliers that 1) are confined to a *feature subspace* but 2) can only be observed within the *global neighborhood* of the outlier, i.e., when the outlier is compared to data points belonging to the same component. This leads to the following definition.

Problem 4.1 (Subspace Outlier in Global Neighborhood) *Given a dataset D over features $\mathcal{F}$ and neighborhood size k, we define the probability p that a data point $d \in D$ is a* subspace outlier in global neighborhood *w.r.t. $F \subseteq \mathcal{F}$ as*

$$p_{F,k}(d) = P(\pi_F(d) \mid \pi_F(NN_k(d))),$$

where $\pi_F(X)$ denotes X projected onto F and $NN_k(d)$ denotes d's global k*-neighborhood, i.e., the k data points closest to d in D (over all features $\mathcal{F}$).*

That is, it is our aim to estimate the probability that a data point is an outlier within a feature subspace, but relative to its neighbors in the complete, global feature space. In the following two subsections we will introduce the concepts and theory needed to accomplish this. Note that we will often drop k from $p_{F,k}$ as this is usually a constant.

Before that, however, it is important to observe that we use the global feature space only to determine a reference collection, after which any subspace can be considered for the actual estimation of the outlier probabilities. Although the *absolute* distances between the data points in $\mathcal{F}$ will be small when the data is high-dimensional, a *ranking* of data points based on distances from a given d is likely to result in neighborhoods that primarily consist of data points belonging to the same component as d. That is, we implicitly assume that the components of the mixture are—to a large extent—separable in the global feature space, but this seems very reasonable for the setting that we consider.

The attentive reader may observe that this problem appears to be contradictory: if a global neighborhood can be reliably selected, then why not simply do global outlier detection? The answer to this question is two-fold:

1. First, the neighborhood of a data point is usually defined as its k-nearest neighbors. Even when the data is high-dimensional and the absolute distances are small, a *ranking* of distances is likely to result in neighborhoods that primarily consist of data points belonging to the same mixture component. (I.e., we implicitly assume that the components of the mixture are distinguishable in the global feature space.)

2. Second, we assume that any strong deviation in a subspace is sufficient to mark a data point as outlier. This is the case in the manufacturing application domain that we study in Section 4.4, in which each (global) steel coil having one or more strong local deviations should be marked as outlier. Such subspace outliers, however, would not be detected by any global approach as they disappear when considering global distances.

In the following two subsections, we first introduce the preliminaries needed for the algorithm that we propose to solve Problem 4.1, after which we introduce the algorithm itself.

4.2.2 Preliminaries

In this subsection we briefly describe LoOP [39] and HiCS [40], as we will build upon both techniques for our own algorithm, which we will introduce in the next section. The main reason for choosing LoOP is that it closely resembles the well-known LOF procedure but normalizes the outlier factors to probabilities, making interpretation much easier. Further, we use an adapted version of the HiCS algorithm to search for relevant subspaces when there is no set of candidate subspaces known in advance.

LoOP [39] Given neighborhood size k and data point d, LoOP computes the probability that d is an outlier. This probability is derived from a so-called *standard distance* from d to reference points S:

$$\sigma(d, S) = \sqrt{\frac{\sum_{s \in S} \text{dist}(d, s)^2}{|S|}}, \tag{4.1}$$

where $\text{dist}(x, y)$ is the distance between x and y given by a distance metric (e.g., Euclidean or Manhattan distance).

Then, the *probabilistic set distance* of a point d to reference points S with 'significance' λ (usually 3, corresponding to 98% confidence) is defined as

$$\text{pdist}(\lambda, d, S) = \lambda * \sigma(d, S). \tag{4.2}$$

From the following step onward nearest neighbors are used as reference sets. That is, given neighborhood size k and significance λ, define the *Probabilistic Local Outlier Factor* (PLOF) of data point d as

$$PLOF_{\lambda,k}(d) = \frac{\text{pdist}(\lambda, d, NN_k(d))}{\mathbf{E}_{s \in NN_k(d)}[\text{pdist}(\lambda, s, NN_k(s))]} - 1. \tag{4.3}$$

Finally, this is used to define Local Outlier Probabilities.

Definition 4.1 (Local Outlier Probability (LoOP)) *Given the previous, the probability that a data point $d \in D$ is a local outlier is defined as:*

$$LoOP_{\lambda,k}(d) = \max\left\{0, \text{erf}\left(\frac{PLOF_{\lambda,k}(d)}{nPLOF \cdot \sqrt{2}}\right)\right\}$$

where $nPLOF = \lambda \cdot Stddev(PLOF)$, i.e., the standard deviation of PLOF values assuming a mean of 0*, and* erf *is the standard* Gauss error function*.*

HiCS [40] HiCS is an algorithm that performs an Apriori-like, bottom-up search for subspaces manifesting a *high contrast*, i.e., subspaces in which the features have high conditional dependences. For a given candidate subspace it randomly selects data slices so that a statistical test can be used to assess whether the features in the subspace are conditionally dependent. To make this procedure robust, this is repeated a number of times (Monte Carlo sampling) and the resulting p-values are averaged. Although the method was originally evaluated using both the Kolmogorov-Smirnov test and Welch's t-test, we here choose the former as this does not require any (parametric) assumptions about the data. Parameters are the number of Monte Carlo samples M (= 50, default value), test statistic size α (= 0.1), and *candidate_cutoff* (= 400), which limits the number of subspace candidates considered.

4.2.3 Global Local Outlier Probabilities

We introduce GLOSS, for *Global–Local Outliers in SubSpaces*, an algorithm for finding local, density-based subspace outliers in global neighborhoods, as defined in Problem 4.1. On a high level, GLOSS, shown in Algorithm 4.1, employs the following procedure. First, if no subspaces are given a subspace search method is used to find suitable subspaces (Line 1). Then, the global k-neighborhood is computed for each data point in the data (2–3). After that, for each data point an outlier probability is computed for each considered subspace, *relative to its global neighborhood* (4–9). Finally, these outlier probabilities are returned as result (10).

As the algorithm computes an outlier probability for each combination of data point and subspace, the probabilities need to be aggregated in order to rank the data points according to outlierness. As we are interested in strong outliers in *any* subspace, we will use the maximum outlier probability found for a data point, i.e., $p(d) = max_{F \in \mathcal{F}}(p_F(d))$. Using the average, for example, would give very low outlier probabilities for data points that [only] strongly deviate in a small subspace.

More in detail, GLOSS builds upon both LoOP and HiCS by integrating both algorithms and adapting them to the global neighborhood setting that we consider in this work.

First, we introduce the *extended standard distance*, inspired by LoOP, which incorporates 1) a feature subspace F and 2) a *global* neighborhood relation G:

$$\sigma(d_F, G_d) = \sqrt{\frac{\sum_{s \in G_d} \text{dist}(d_F, s_F)^2}{|G_d|}}, \tag{4.4}$$

where d_F and s_F are shortcuts for $\pi_F(d)$ and $\pi_F(s)$ respectively, and G_d is the global neighborhood defined as $G_d = NN_k(d)$.

Then, using *probabilistic set distance* as defined in the previous subsection together with the extended standard distance, we define the *Probabilistic Global*

Local Outlier Factor PGLOF as:

$$PGLOF_{\lambda,G_d}(d_F) = \frac{\text{pdist}(\lambda, d_F, G_d)}{\mathbf{E}_{s\in G_d}[\text{pdist}(\lambda, s, G_s)]} - 1 \tag{4.5}$$

Finally, a subspace outlier probability $p_{F,k}(d)$ is computed for each data point and subspace according to Definition 4.1, but using *PGLOF* instead of *PLOF*; see Line 9 of Algorithm 4.1. That is, with the *global* neighborhood projected onto the features in the selected *subspace*.

Algorithm 4.1 GLOSS

Given: Dataset D, neighborhood size k, optional: subspaces $\mathcal{F}$
1: $\mathcal{F} = \text{SubspaceSearch}(D)$ {O}nly if $\mathcal{F}$ not given
2: **for all** $d \in D$ **do**
3: $\quad G_d = NN_k(d)$
4: **end for**
5: **for all** $d \in D$ **do**
6: $\quad$ **for all** $F \in \mathcal{F}$ **do**
7: $\quad\quad \sigma(d_F, G_d) = \sqrt{\frac{\sum_{s\in G_d} \text{dist}(d_F, s_F)^2}{|G_d|}}$
8: $\quad\quad \text{pdist}(\lambda, d_F, G_d) = \lambda \cdot \sigma(d_F, G_d)$
9: $\quad\quad PGLOF_{\lambda,G_d}(d_F) = \frac{\text{pdist}(\lambda, d_F, G_d)}{\mathbf{E}_{s\in G_d}[\text{pdist}(\lambda, s, G_s)]} - 1$
10: $\quad\quad p_{F,k}(d) = \max\left\{0, \text{erf}\left(\frac{PGLOF_{\lambda,G_d}(d_F)}{nPGLOF\cdot\sqrt{2}}\right)\right\}$
11: $\quad$ **end for**
12: **end for**
13: **return** p

4.2.4 Subspace Search

GLOSS can either perform subspace search or use a given set of relevant subspaces. In the latter case, the subspace search (Line 1 in Algorithm 4.1) is skipped. By parameterizing this, we allow background knowledge to be used to reduce the number of subspaces whenever possible, hence avoiding an exponential search for subspaces and thus reducing runtime. In the manufacturing case study that we will present in Section 4.4, for example, there is a natural collection of subspaces that can be exploited.

When subspace search is enabled, the search procedure of HiCS is used. However, instead of testing each feature of a candidate subspace against the remaining subspace features, Gloss tests each candidate subspace feature against the remainder of the *entire feature space*, emphasizing the relation between local and global spaces. As such, the algorithm searches for subspaces that exhibit high contrast relative to the global feature space. Because subspace search is adapted from HiCS, the parameters and their default values are the same as those described in subsection 4.2.2.

The maximum number of interesting subspaces for HiCS is usually set to 100, also for Gloss this is generally a good value though less is most of the time fine as well, depending on the dataset.

4.3 Experiments

We evaluate Gloss on 1) synthetic data, 2) benchmark data with implanted outliers, 3) benchmark data with the minority class as outlier class, and 4) a real-world dataset provided by an industrial partner. The source code and experimental setup can also be found on the Github repository [52].

In the first experiment, in Subsection 4.3.1, we simulate an (unbalanced) Boolean classification task where the class labels are 1) outlier and 2) not an outlier. This is a very common approach in outlier detection, because objective evaluation is very hard otherwise. Performance is quantified by 1) *Area Under the Curve* (AUC) of the ROC curve and 2) runtime.

We compare Gloss to LoOP, LOF, HiCS, and *LoOP local*, a variant of LoOP that detects outliers in each 2D subspace and then assigns the maximum probability over all subspaces to the data point. For all algorithms the neighborhood size k is set to 20, which is considered to be sufficiently large; the distance metric is set to Euclidean. For both HiCS and Gloss, the parameters are set to their defaults and the maximum number of subspaces considered is also set to the default: 100.

4.3.1 Synthetic Data

Setup We first devise a generative model to generate data with known outliers that satisfy the assumptions of our problem statement: the data is a mixture of samples from different distributions, and outliers have values sampled from another distribution for some random subspace. More formally, the generative process generates a dataset D with features $\mathcal{F}$ and clusters C, where each cluster $c \in C$ is assigned a random center μ_c and variance ${\sigma^2}_c$. Each data point $d \in D$ is assigned to one of the clusters uniformly at random, denoted $C(d)$, and then sampled from a normal distribution with specified center and variance:

$$\forall d \in D : d \leftarrow \mathcal{N}(\mu_{C(d)}, {\sigma^2}_{C(d)}).$$

After generating the mixed dataset, outliers O are introduced by changing a random subset of the features for some of the data points. Given a data point o, a random $F \subset \mathcal{F}$ and a randomly chosen cluster $r \neq C(o)$, o is marked as outlier and o projected onto F is changed as follows:

$$o_F \leftarrow \mathcal{N}(\mu_r, {\sigma^2}_r)_F.$$

Experiments are performed on synthetic datasets with 1000 data points, of which 50 are marked as outliers. The number of dimensions d is set to $10, 20, 50, 100, 200$ or 400; the number of clusters tested are $2, 3$ and 5; and μ is per dimension randomly drawn from $[0, 2]$, $[0, 3]$, $[0, 5]$ or $[0, 10]$ (${\sigma^2}_c$ is fixed to 1). This results in 18 parameter settings per dimensionality.

Results Figure 4.2 shows ROC curves for all algorithms per dimensionality, using 3 clusters and μ drawn from $[0, 3]$. Table 4.1 presents the obtained AUC scores and run times averaged over all 18 runs per dimensionality. It can be observed from Table 4.1 that the purely local subspace analysis done by Local LoOP completely fails to identify the 'hidden' outliers, whereas HiCS and the global outlier detection methods fail when the number of dimensions increases. Gloss, on the other hand, is able to detect most outliers even when the dimensionality increases all the way up to 400. From the ROC curves in Figure 4.2 it can be observed that

GLOSS tends to find many more outliers at very low false positive rates, while other algorithms only manage to catch up once the false positives rate increases substantially. From the results per individual parameter setting (not included here but available on GitHub [52]), we can see that a higher μ makes it easier for all algorithms to detect the outliers. This makes sense, since the clusters become more separated and therefore the impact of the local deviation on the global space will be higher. The number of clusters in the data does not seem to be of substantial importance.

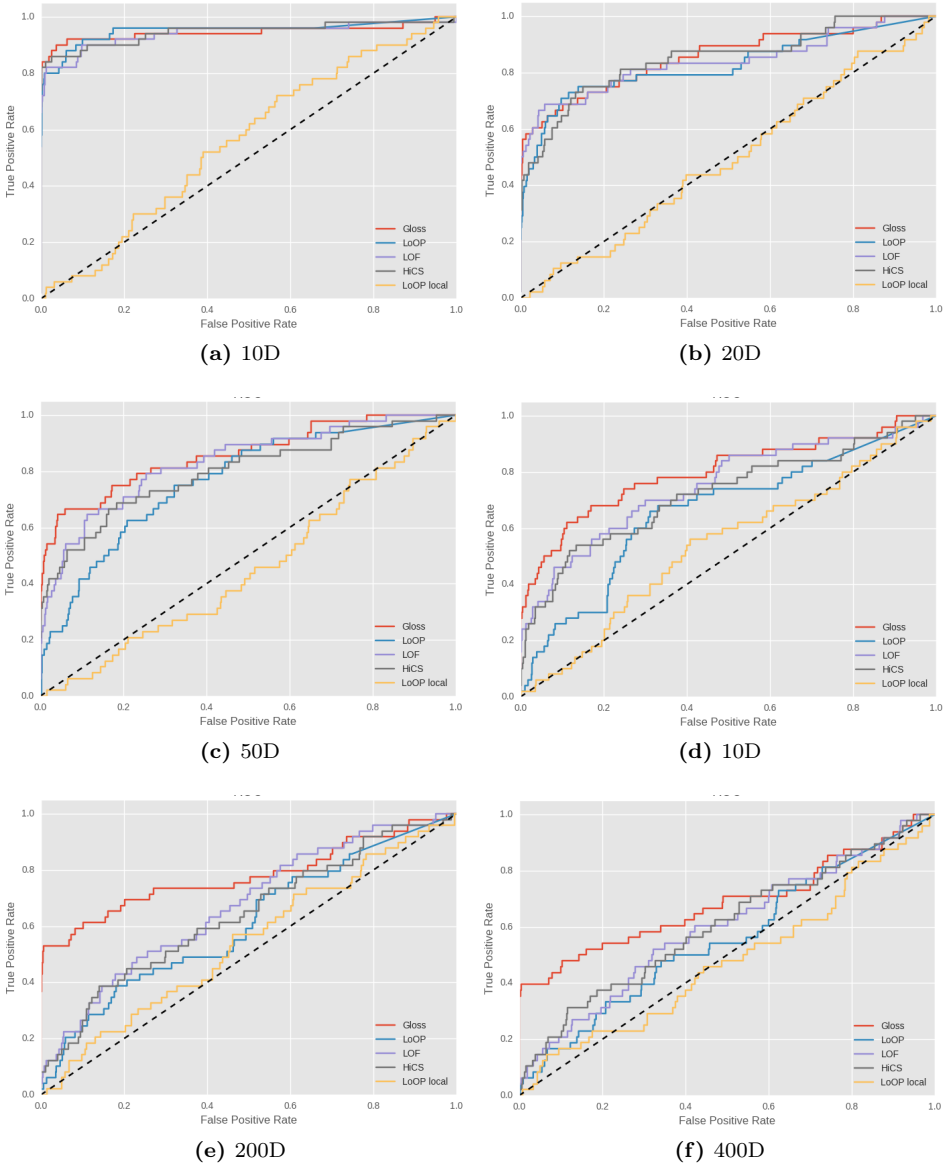
Table 4.1: Results on synthetic data. Average AUC and runtime in seconds for each algorithm, per dimensionality, averaged over all other parameter settings.

	AUC					Runtime				
d	GLOSS	HiCS	LOF	LoOP	L.LoOP	GLOSS	HiCS	LOF	LoOP	L.LoOP
10	0.955	**0.96**	0.96	0.96	0.55	2.3	3.2	0.1	0.4	1.9
20	**0.95**	0.94	0.94	0.94	0.53	4.0	3.5	0.1	0.4	3.4
50	**0.94**	0.92	0.92	0.90	0.51	12.2	9.3	0.2	0.6	8.7
100	**0.93**	0.90	0.90	0.85	0.54	30.0	41.1	0.3	0.9	17.7
200	**0.92**	0.85	0.87	0.80	0.52	79.4	91.5	0.6	1.7	38.6
400	**0.90**	0.81	0.84	0.73	0.48	225.9	232.8	1.2	2.4	57.6

4.3.2 Benchmark Data with Implanted Outliers

Setup Next we compare GLOSS to its competitors using a large set of well-known benchmark data from the *UCI machine learning repository* [25]: Ann Thyroid, Arrhythmia, Glass, Diabetes, Ionosphere, Pen Digits 16, Segments, Ailerons, Pol, Waveform 5000, Mfeat Fourier and Optdigits.

Previous papers usually considered the minority class as 'outlier class' for purposes of evaluation, but this clearly would not demonstrate the strengths of our approach: we assume the data to be a mixture of components (i.e., classes), and we search for outliers *within* those classes. We therefore use the UCI datasets as examples of realistic data and implant artificial outliers. That is, we pick a random sample of 10% of the data points and transform each such data point to an outlier by replacing a randomly picked subspace with the values of a data point from a different class (the size of each subspace was chosen uniformly from



(a) 10D **(b)** 20D

(c) 50D **(d)** 10D

(e) 200D **(f)** 400D

Figure 4.2: Results on synthetic data. ROC curves for each algorithm and per dimensionality, with μ randomly drawn from $[0, 3]$ and 2 clusters per dataset.

$[2, max(2, 0.1 * d)]$). Note that the datasets most likely already contain 'natural' outliers, which makes the task at hand even more difficult.

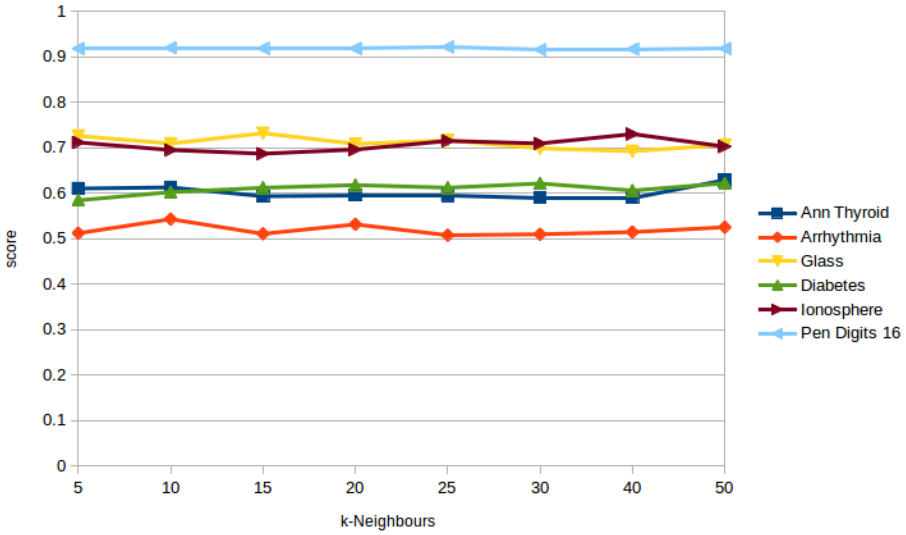


Figure 4.3: GLOSS accuracy for benchmark datasets with increasing neighborhood sizes and 5% implanted outliers.

Results Figure 4.3 shows the effect of the neighborhood size on the performance of GLOSS. It can be seen that the method is very robust with respect to this setting. As $k = 20$ is considered to be a "good" choice in literature for LoOP and related local outlier detection methods, we chose this as the default for all other experiments and methods.

Table 4.2 presents average AUC scores and running times over ten runs per dataset, together with basic dataset properties, for all competing methods. GLOSS clearly outperforms its competitors for most datasets when it comes to AUC and is about as fast as HiCS. From this we can conclude that it is beneficial to use GLOSS when the data consists of multiple components and outliers may be hidden as a result of that.

Table 4.2: Results on benchmark data with implanted outliers. AUC scores and running times in seconds for each algorithm.

			AUC					Runtime				
Dataset	$\|D\|$	d	GLOSS	HiCS	LOF	LoOP	L.LoOP	GLOSS	HiCS	LOF	LoOP	L.LoOP
Ann Thyroid	3772	30	**0.61**	0.57	0.60	0.59	0.55	115.8	34.8	0.3	1.2	20.1
Arrhythmia	452	279	**0.54**	0.53	0.51	0.53	0.53	327.8	1517.8	0.1	0.4	22.4
Glass	214	9	**0.73**	0.68	0.71	0.71	0.55	20.5	14.4	0.0	0.1	0.3
Diabetes	768	8	**0.62**	0.59	0.60	**0.62**	0.51	25.1	10.1	0.0	0.1	0.9
Ionosphere	351	34	**0.70**	0.61	0.64	0.61	0.57	17.6	13.1	0.0	0.1	2.0
Pen Digits 16	10692	16	**0.92**	0.86	0.92	0.91	0.50	236.9	53.4	1.5	3.0	28.7
Segments	2310	20	**0.82**	0.80	0.77	0.80	0.75	61.8	15.2	0.1	0.5	7.6
Ailerons	13750	41	**0.69**	0.61	0.51	0.65	0.61	396.8	91.9	0.7	8.2	109.7
Pol	15000	49	0.58	0.54	**0.59**	0.58	0.51	475.1	256.4	4.9	7.6	168.4
Waveform 5000	5000	41	0.57	0.57	**0.58**	0.57	0.52	172.1	76.5	2.6	2.7	34.8
Mfeat Fourier	2000	77	**0.63**	0.54	0.59	0.57	0.51	84.3	53.7	0.7	1.2	29.2
Optdigits	5620	65	**0.69**	0.58	0.68	0.66	0.52	209.5	117.7	4.9	4.6	73.0
Average			**0.67**	0.62	0.64	0.65	0.55	174.1	176.6	1.2	2.3	38.7

4.3.3 Benchmark Data with Minority Class as Outliers

Setup We do not expect using the minority class of a dataset as outlier class to demonstrate the strengths of our approach. Nevertheless, we do not want our improved algorithm to perform worse on the regular local outlier detection task either. Hence, we also compare GLOSS to its competitors using the same benchmark datasets but with outliers defined by the more usual procedure of using the minority class as 'outlier class'. Apart from that, we use the same setup and parameters as in Subsection 4.3.2.

Results Table 4.3 presents the average AUC scores obtained over ten runs per dataset. The results show that GLOSS performs pretty much on par with the state-of-the-art, demonstrating that our proposed method is capable of detecting 'regular' outliers as well as the ones that GLOSS identifies but other methods miss (see previous subsections).

4.4 Case Study: Outlier Detection for BMW

The last series of experiments of this section are performed on a proprietary dataset made available by the *BMW Group* at plant Regensburg. This dataset

Table 4.3: Results on benchmark data using the minority class as outliers. AUC scores for each algorithm.

Dataset	GLOSS	HiCS	LOF	LoOP	L.LoOP
Ann Thyroid	0.759	0.581	0.727	0.779	**0.889**
Arrhythmia	0.581	**0.646**	0.48	0.617	0.582
Glass	0.771	**0.818**	0.815	0.744	0.621
Diabetes	**0.575**	0.512	0.495	0.566	0.508
Ionosphere	0.886	**0.921**	0.881	0.881	0.733
Pen Digits 16	0.473	0.522	0.461	0.465	**0.524**
Segments	0.522	0.493	0.512	0.52	**0.530**
Ailerons	0.839	0.977	0.185	0.634	**0.987**
Pol	0.445	0.461	0.439	0.434	0.467
Waveform 5000	0.503	0.496	0.498	0.5	**0.512**
Mfeat Fourier	0.442	0.518	0.487	0.439	0.5
Optdigits	0.519	**0.57**	0.538	0.545	0.483
Average	0.61	**0.626**	0.543	0.594	0.611

was one the motivations for this work: the data is high-dimensional and a mixture of different, unknown components. Moreover, it is essential for BMW to be able to identify any outliers in the data, as this directly influences their car manufacturing process.

The data concerns steel coils, which is the raw material used as input at the stamping plant (also called 'press shop'). Before entering the stamping process, each coil—of 2–3 km long—is unrolled and cut into shorter pieces. During this process, a large number of measurements is made. We aim to use these measurements to detect steel coils that strongly deviate from a typical coil *in some specific region*. A complicating factor is that the data contains *measurements for different types of steel from different suppliers*, but this important information is not available in the data. Hence, we are dealing with mixed data and we are thus facing exactly the problem formalized as Problem 4.1, for which we proposed GLOSS as solution.

Setup The dataset, containing all measurements done from December 2014 to December 2015, consists of 2204 data points and has 1200 dimensions, grouped into 100 12-dimensional subspaces using the spatial aspects of the data. Each data point represents a coil having 100 segments (in length) and 3 tracks (in

width). The most important measurements [8], and the ones we use, are *IMPOC*, quantifying magnetic properties of the steel, and *Oil levels*, quantifying the amount of oil on the coil. Each subspace consists of 3 IMPOC and 9 Oil level values averaged over a segment of size 2% of the length of the coil; the 100 subspaces are consecutive, overlapping segments covering the entire coil.

We compare GLOSS to LoOP using all global features and to Local LoOP ran on each of the 100 individual segments/subspaces. Other algorithms are not included in the evaluation because of the high-dimensionality of the data; run times would be unreasonably long.

Results As expected, LoOP is unable to detect local outliers: it does not take advantage of the spatial information and cannot deal with the very large number (1200) of dimensions. The results obtained by GLOSS and our Local LoOP variant are generally similar, but are substantially—and importantly—different for some of the steel coils, as we will show in detail shortly. Moreover, Local LoOP is slower than GLOSS, since the neighborhood of a coil needs to be computed for each individual subspace, whereas GLOSS only needs to compute a global neighborhood once.

We now zoom in on the 512 coils recorded in March 2015, a representative month. By focusing on data from a specific month, we simulate the setting in which the stamping plant operator will inspect the results in the future; GLOSS is currently being implemented in the production environment at BMW. Given that deviations in the steel coils directly influence the manufacturing process, this is expected to improve the stability of the process and the quality of the products.

When comparing the outlier rankings obtained with GLOSS and Local LoOP for this particular month, we observe that many top outliers appear in high positions in both rankings. However, 1) some coils are ranked very differently by the two approaches and 2) GLOSS ranks some coils as outliers that Local LoOP does not. One such a coil is depicted in Figure 4.4, showing both the outlier probabilities computed by both methods, and the IMPOC and Oil level measurements. While GLOSS ranks this coil 5th, Local LoOP ranks it 103th. Clearly an operator would inspect this coil, labeled $B1$, if GLOSS were used to rank the coils, but not if Local LoOP would have been used. We asked a domain expert to inspect the

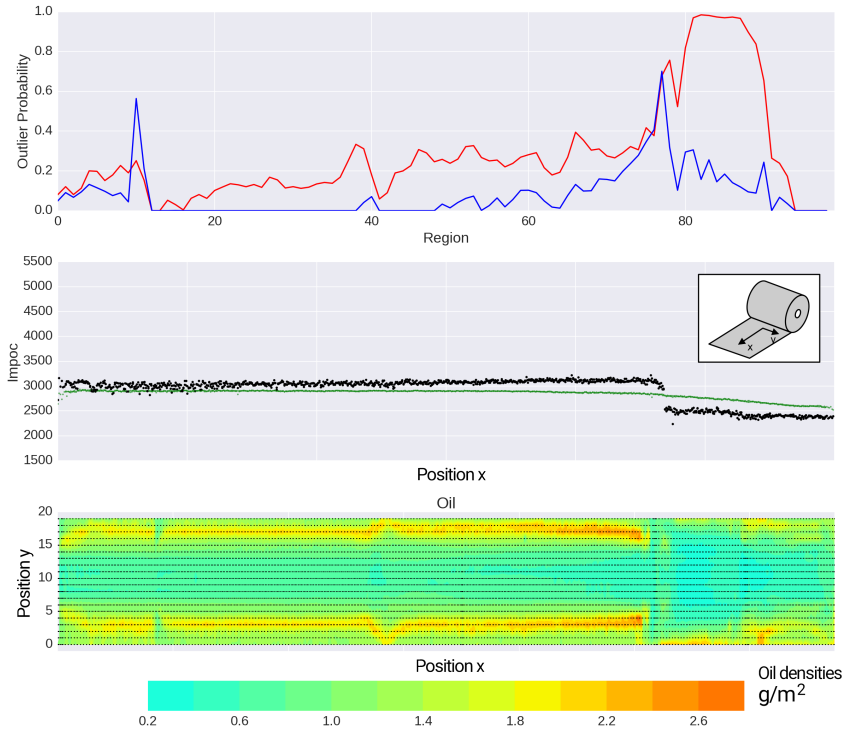


Figure 4.4: Results for coil $B1$. Top: GLOSS (red line) and LoOP (blue line) outlier probabilities for each of 100 consecutive coil segments. Middle: IMPOC measurements over the whole length of the coil, both for this particular coil (black) and averaged over its 20 global neighbors (green). Bottom: Oil level measurements visualized in 2D, representing the entire surface of the coil.

measurements and outlier probabilities of this coil and others. He reported back to us that the probabilities computed using GLOSS more accurately reflect the extend to which the coils are outliers.

More in detail, the low oil levels near the far right of the coils together with the sudden drop in IMPOC are clear deviations, as correctly detected by GLOSS. Local LoOP, however, reports this as normal behaviour since similar oil levels occur *in the same segments of other coil types* and only detects the small areas where the measurements change rapidly. In Figure 4.4, LoOP only detects an outlier because the transition from low to high oil levels happens to be located

Table 4.4: Case study. Outlier rankings obtained by Local LoOP, GLOSS, and BMW domain expert.

Rank	L.LoOP	GLOSS	BMW Expert
1	A1	B1	B1
2	A2	B2	B10
3	A3	B3	B6
4	A4	B4	B3
5	A5	B5	A6
6	A6	B6	A1
7	A7	B7	B8
8	A8	B8	A10
9	A9	B9	A4
10	A10	B10	B4

such that it deviates from its local neighbors. However, when looking at globally similar coils one observes that the high oil levels in the middle of the coil are very rare, which explains the high outlier probabilities given by GLOSS.

Next, to further validate the rankings provided by our method, a domain expert of BMW was shown two top-10 outlier coil rankings, one obtained by GLOSS and one by Local LoOP (without duplicates; a coil was left out from a ranking if it was ranked higher by the other method). Of course, the test was blind, i.e., the domain expert did not know which method generated which ranking. For each coil in either top-10, the domain expert was shown the plots as in Figure 4.4, but only with the outlier probabilities for the corresponding method. Given the two rankings and plots, the domain expert was asked to rank the 20 (unique) coils according to the perceived degree of outlierness from the domain perspective. Table 4.4 shows the labels for the coils in the top-10 rankings of Local LoOP and GLOSS, plus the ranking given by the domain expert (using these labels). It is striking that the top four coils selected by the domain expert were all selected by GLOSS, with the top ranked coil being the same coil as the top ranked coil identified by GLOSS. This confirms that our proposed algorithm is capable of detecting and ranking important outliers that existing algorithms overlook.

For the application at our industrial partner, deviations in the measurements often indicate problems with the material and these may cause problems during the manufacturing process. Per year, over 100 000 coils are processed at this plant,

making it infeasible for operators to inspect every single coil. Thus, GLOSS will help to narrow this down by providing outlier rankings and probabilities. GLOSS is able to find these hidden anomalies while LoOP both on the global space as on the local subspaces is not always able to.

While most of the results show similar outcomes for both multiple times LoOP and GLOSS, there are quite a few cases where GLOSS would classify a segment as outlier and LoOP would not.

4.5 Conclusions and Outlook

An algorithm, GLOSS, is proposed to find local subspace outliers in high-dimensional mixed datasets. The algorithm is evaluated on synthetic high-dimensional data, open access datasets from the UCI machine learning repository and a closed dataset from BMW Regensburg.

It can be seen from the results shown in Section 4.3 that the proposed algorithm, GLOSS, outperforms existing methods on the synthetic data with hidden subspace outliers once the dimensionality grows large. From the results it can be observed that HiCS has a relative high execution time once the number of dimensions grows, this is due to the search for interesting subspaces, even though this is already limited by $4 \cdot D$. The proposed algorithm as well as Local LoOP, takes only two dimensional subspaces into account. Given the global neighborhood, there is less need to consider every possible subspace. However, a search for these subspaces using HiCS' method or another approach would most likely improve accuracy. The results for the open datasets show that our algorithm also performs well on real world datasets. The real world case on BMW data, shows us that GLOSS is capable of finding some important outliers that would otherwise remain undetected by using existing algorithms, confirming that one should keep an eye on the global perspective even when doing local outlier detection.

For future research it would be very interesting to look into efficient relevant subspace detection algorithms to improve the accuracy of GLOSS without significantly increasing complexity. Another improvement would be to use locality

sensitive hashing to define the neighborhood of each point, which would provide a much faster solution than using standard kNN.

5

Cluster Kriging

Data driven models and model-driven optimization are two of the most important parts of the data driven framework (Section 2.3). *Kriging* or *Gaussian Process Regression* is a popular kernel based data driven regression model. Kriging is capable of achieving high accuracy predictions and providing, in addition, the predicted variance, also known as the Kriging variance. The Kriging variance is used in many applications as the uncertainty or quality measure of a prediction and is heavily exploited in surrogate model-based optimization.

However, the computational and space complexity of Kriging, that is cubic and quadratic in the number of data points respectively, is a major bottleneck, especially with the quantities of data available for the PROMIMOOC project. To address these issues, Cluster Kriging is proposed as a solution for the time and space complexity bottleneck. In addition, Cluster Kriging can be used as the surrogate model for *Efficient Global Optimization* to allow model-driven optimization on big data.

In this chapter, a general methodology for the complexity reduction of Kriging, called *Cluster Kriging*, is proposed. The main principle of Cluster Kriging is that the whole data set is partitioned into smaller clusters and multiple Kriging models are built on top of them. In addition, four Kriging approximation algorithms are proposed as candidate algorithms within the new framework. Each of these algorithms can be applied to much larger data sets while maintaining the advantages and power of Kriging. The proposed algorithms are explained in detail and compared empirically against a broad set of existing state-of-the-art Kriging approximation methods on a well-defined testing framework. It is also shown that Cluster Kriging can be used with the popular model-based optimization framework; Efficient Global Optimization. According to the empirical study, the proposed algorithms consistently outperform the existing algorithms. Moreover, some practical suggestions are provided for using the proposed algorithms.
The content of this chapter is primarily based on the publications [53, 54, 55, 56].

5.1 Introduction

Kriging, or *Gaussian Process Regression* [57] is a popular and elegant kernel based regression model capable of modeling very complex functions. Kriging is used in many fields e.g., engineering, mining and geology, as a tool for the analysis of data sets, for prediction purposes and for *Surrogate model-based optimization* [58]. Many other regression models exist, such as parametric models, which are easy to interpret but may lack expressive power to model complex functions. On the other hand, *Regression Tree* based methods like *Random Forests* [59] or *Gradient Boosted Decision Trees* lack the advantage of interpretation [60] but have more expressive power. Another method is *Linear Model Trees* [61], which uses a tree structure with linear models at the leaves of the tree. There are also more complex algorithms like *Neural Networks*, or *Extreme Learning Machines* [62], that are able to model very complex functions but are usually not easy to work with in practice. There are also different kernel based methods such as *Support Vector Machines* [63] and *Radial Basis Functions* [64]. The main advantage of Kriging over other regression methods is that Kriging provides not only the estimate of the value of a function, but also the mean squared error of the estimation, the so-called *Kriging variance*. The Kriging variance can be seen as the uncertainty assessment of the model and has been exploited in surrogate model based optimization and many other applications. Despite the clear advantage of the Kriging variance, Kriging suffers from one major problem, the high training time and space complexity, which are $O(n^3)$ and $O(n^2)$, respectively. Where n denotes the number of points. To overcome this complexity problem, Kriging approximation algorithms such as [65] and [66] are introduced. Unfortunately, these approximation algorithms are usually less accurate than the original Kriging algorithm.

An overview of Kriging approximation methods is presented and a novel divide and conquer based approach, *Cluster Kriging* (CK), is introduced. The novel Cluster Kriging framework contains three steps, Partitioning, Modeling and Predicting. Each of the steps can be implemented using a wide range of approaches, which are explained in detail in this chapter. Using these approaches four algorithms are implemented and compared against each other and the state of the art. One particular interesting and novel algorithm that uses the Cluster Kriging

methodology is the proposed **Model Tree Cluster Kriging** (MTCK). MTCK uses a regression tree with a specified number of leaf nodes to partition the data in the objective space. A Kriging model is then built on each partition defined by the tree's leaves. MTCK uses only one of the trained Kriging models per unseen record to predict, depending on which leaf node the unseen record is assigned to. The proposed algorithms are evaluated and compared to several state-of-the-art alternative Kriging approximation algorithms. A well-defined testing framework for Kriging approximation algorithms [67] is adopted for the comparisons.

In addition to the Cluster Kriging varients proposed, it is also shown that Cluster Kriging can be integrated in the Efficient Global Optimization framework. Enabling efficient global optimization for large data sets.

5.2 Kriging

Loosely speaking, Kriging is a stochastic interpolation method in which the output value of a stochastic process is predicted as a linear function of the observed output values [68, 69]. In particular, Kriging is the best linear unbiased predictor (BLUP) and the corresponding mean squared error of prediction is used for uncertainty qualification. Kriging originates from the field of spatial analysis/geostatistics and more recently is being widely used in Bayesian optimization and design and analysis of computer experiments (DACE) [70, 71]. The model features in providing the theoretical uncertainty measurement of estimations.

When the stochastic process is assumed to be Gaussian, Kriging is equivalent to Gaussian Process Regression (GPR), where the *posterior* distribution of the regression function (posterior Gaussian process) is inferred through Bayesian statistics. In this chapter, we shall consider this special case and adopt the mathematical treatment of the Gaussian process. Assume that input data points are summarized in the set $\mathcal{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}\} \subseteq \mathbb{R}^d$ and the corresponding output variables are represented as $\mathbf{y} = [y(\mathbf{x}^{(1)}), y(\mathbf{x}^{(2)}), \ldots, y(\mathbf{x}^{(n)})]^\top$. Specifically, the mostly used variant of Kriging, *Ordinary Kriging*, models the regression function f as a random process, that is a combination of an *unknown* constant trend μ

with a centered Gaussian Process ε. The output variables are considered as the "noisy" observation of f, that is perturbed by a Gaussian random noise γ:

$$y(\mathbf{x}) = f(\mathbf{x}) + \gamma(\mathbf{x}) = \mu + \varepsilon(\mathbf{x}) + \gamma(\mathbf{x}),$$

$$\varepsilon(\mathbf{x}) \sim \mathcal{N}(0, \sigma_\varepsilon^2(\mathbf{x})), \quad \gamma(\mathbf{x}) \sim \mathcal{N}(0, \sigma_\gamma^2), \quad \varepsilon \perp\!\!\!\perp \gamma.$$

Note that the noises (error terms) γ are assumed to be homoscedastic (identically distributed) and independent, both from each other and the Gaussian Process ε. The centered Gaussian process ε is a stochastic process which possesses zero mean everywhere and any finite collection of its random variables has a joint Gaussian distribution [57]. It can be completely specified by providing a covariance function $k(\cdot, \cdot)$ to calculate the pairwise covariance:

$$k(\mathbf{x}, \mathbf{x}') = \mathrm{Cov}[\varepsilon(\mathbf{x}), \varepsilon(\mathbf{x}')].$$

The covariance function $k(\cdot, \cdot)$ is a kernel function performing the so-called "kernel trick", which computes the inner product on the feature space as a function in the input space. Consequently, the variance $\sigma_\varepsilon^2(\mathbf{x})$ of a Gaussian process ε is independent from the input $\mathbf{x}$ and thus denoted as σ_ε^2 in the following. In practice, a common choice is the Gaussian covariance function (also known as squared exponential kernel):

$$k(\mathbf{x}, \mathbf{x}') = \sigma_\varepsilon^2 \prod_{i=1}^{d} \exp\left(-\theta_i (x_i - x_i')^2\right), \tag{5.1}$$

where θ_i's are called hyper-parameters, that are either predetermined or estimated through model fitting, and σ_ε^2 is inferred by the maximum likelihood method. By using the Gaussian kernel, the resulting Gaussian process is stationary in the sense that the process variance is constant: $\forall \mathbf{x} \in \mathbb{R}^d, k(\mathbf{x}, \mathbf{x}) = \sigma_\varepsilon^2$.

To infer output value $y^{(t)} = y(\mathbf{x}^{(t)})$ at an unobserved data point $\mathbf{x}^{(t)}$, the joint distribution of $y^{(t)}$ and observed outputs $\mathbf{y}$ are derived, conditioning on the input data set $\mathcal{X}$, $\mathbf{x}^{(t)}$ and the unknown prior mean μ. Such a joint distribution is a

multivariate Gaussian and is expressed as follows;

$$\begin{bmatrix} y^{(t)} \\ \mathbf{y} \end{bmatrix} \mid \mathcal{X} \sim \mathcal{N}\left(\mu \mathbf{1}_{n+1}, \begin{bmatrix} \sigma_\varepsilon^2 + \sigma_\gamma^2 & \mathbf{c}^\top \\ \mathbf{c} & \mathbf{\Sigma} + \sigma_\gamma^2 \mathbf{I} \end{bmatrix}\right), \tag{5.2}$$

$$\mathbf{c}_i = k(\mathbf{x}^{(t)}, \mathbf{x}^{(i)}), \quad \mathbf{\Sigma}_{ij} = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}),$$

where $\mathbf{1}_{n+1}$ denotes a column vector of length $n+1$ that contains only 1's. The homogeneous variance σ_γ^2 of the noise can be either determined by the user or estimated through maximum likelihood method. The *posterior* distribution of $y^{(t)}$ can be calculated by marginalizing μ out and conditioning on the observed output variables $\mathbf{y}$ [57]. Without any derivations, the posterior distribution for Ordinary Kriging is again Gaussian [72]:

$$y^{(t)} \mid \mathcal{X}, \mathbf{y}, \mathbf{x}^{(t)} \sim \mathcal{N}\left(m(\mathbf{x}^{(t)}), s^2(\mathbf{x}^{(t)})\right) \tag{5.3}$$

where the posterior mean and variance are expressed as:

$$m(\mathbf{x}^{(t)}) = \hat{\mu} + \mathbf{c}^\top \left(\mathbf{\Sigma} + \sigma_\gamma^2 \mathbf{I}\right)^{-1} (\mathbf{y} - \hat{\mu}\mathbf{1}_n) \tag{5.4}$$

$$\begin{aligned} s^2(\mathbf{x}^{(t)}) = \sigma_\gamma^2 + \sigma_\varepsilon^2 - \mathbf{c}^\top \left(\mathbf{\Sigma} + \sigma_\gamma^2 \mathbf{I}\right)^{-1} \mathbf{c} \\ + \frac{(1 - \mathbf{c}^\top \left(\mathbf{\Sigma} + \sigma_\gamma^2 \mathbf{I}\right)^{-1} \mathbf{1}_n)^2}{\mathbf{1}_n^\top \left(\mathbf{\Sigma} + \sigma_\gamma^2 \mathbf{I}\right)^{-1} \mathbf{1}_n} \end{aligned} \tag{5.5}$$

$$\hat{\mu} = \frac{\mathbf{1}_n^\top \left(\mathbf{\Sigma} + \sigma_\gamma^2 \mathbf{I}\right)^{-1} \mathbf{y}}{\mathbf{1}_n^\top \left(\mathbf{\Sigma} + \sigma_\gamma^2 \mathbf{I}\right)^{-1} \mathbf{1}_n}$$

Note that the estimation of the trend, $\hat{\mu}$ is obtained by maximum a posteriori principle (MAP). The posterior mean function (Eq. 5.4) is used as the predictor while the posterior variance (Eq. 5.5) is the so-called *Kriging variance* that measures the uncertainty of the prediction.

5.3 Relevant Research

Despite the theoretically sound development of the Kriging model, it suffers several issues when applied to large data sets. The major bottleneck is the high time

and memory complexity of the model fitting process: The inverse of the covariance matrix $\mathbf{\Sigma}^{-1}$ needs to be computed for both the posterior mean and variance (Eq. 5.4 and 5.5), which has roughly $O(n^3)$ time complexity[1]. Moreover, when optimizing the hyper-parameters of the kernel function, the log likelihood function of those parameters is again calculated through $\mathbf{\Sigma}^{-1}$, resulting in a $O(n^3)$ computational cost per each optimization iteration. Thus, for a large data set, such a high overhead in model fitting renders Kriging inapplicable in practice. Various attempts have been made to overcome the computational complexity issue of Kriging [57]. The contributions towards solving this issue can be roughly split into three categories:

Subset Methods

The first category of approximation algorithms uses only a subset of the complete data set to approximate a full Kriging model. The idea behind these methods is to get a realistic representation of the complete data set by taking only a small portion of the data points. The main issue with these subset approximation algorithms is how to identify a subset that represents the complete data set.

Subset of Data (SoD) [73] is a naive approach in reducing complexity by taking a subset of $m < n$ data points. The points are ussualy taken at random. The obvious disadvantage of such an approach is that possible valuable information is lost in the process. Taking a representative subset of data points is a non-trivial task.

Subset of Regressors (SoR) [74] approximates Kriging by a linear combination of kernel functions on a set of basis points. The basis points are linearly weighted to construct the predictor. The choice of the basis points *does* influence the final outcome. As noted also in [75], there are only m (number of basis points) degrees of freedom in the model because the model is degenerate (finite linear-in-the-parameters), which might be too restrictive.

[1] There are asymptotically faster algorithms for matrix inversion, e.g., Strasssen's $O(n^{2.807})$ and Stothers $O(n^{2.373})$, but their practical performance is worse than some methods with $o(n^3)$ time complexity.

Approximation using Sparsity

The second category of approximation algorithms approximate the covariance matrix using sparsity based methods. Most of these algorithms also use a (relevant) subset of the data like in the category mentioned above.

Sparse On-Line Gaussian Processes (OGP) [76] uses a Bayesian on-line algorithm, together with a sequential construction of a subsample of the data that specifies the prediction of the GP model. The idea behind constructing a subsample of basis vectors is very similar to The Fully Independent Training Conditional mentioned next. The advantage of OGP is that additional data points can be added to the OGP model without always completely retraining the model.

Fast Kriging with Gaussian Markov Random Fields [66] is an algorithm that uses an approximation of the covariance matrix with a sparse precision matrix. It uses *Gaussian Markov Random Fields* (GMRF) on a reasonable dense grid to exploit the computational benefits of a Markov field while keeping the formula of Kriging weights. This method reduces the complexity for simple and ordinary Kriging, but might not always be efficient with universal Kriging.

The Fully Independent Training Conditional (FITC) [77, 78]. Snelson and Ghahramani proposed what they called Sparse Gaussian Processes using Pseudo-inputs. It uses a more sophisticated likelihood approximation with a richer covariance. It is a non-degenerate version of the *SoR* algorithm. By providing a set of basis points (Pseudo inputs), the model is fitted and validated on the training data. As with *SoR* the choice of basis points is a problem, this is usually either a subset of the training data or a uniform distribution over the input space.

Divide and Conquer Methods

The last category contains methods that divide a (big) data set into several smaller data sets and build a model for each of them. How to split the data set into

smaller data sets and how to combine the different models is what makes these algorithms unique. The proposed Cluster Kriging algorithms also belong to this category.

Bayesian Committee Machines (BCM) [65] is an algorithm similar to the one we propose, but developed from a completely different perspective. The basic motivation is to divide a huge training set into several relatively small subsets and then construct Kriging models on each subset. The benefit of this approach is that the training time on each subset is satisfactory and the training task can be easily parallelized. After training, the prediction is made by a weighted combination of estimations from all the Kriging models. BCM uses batch prediction to speed up the computation even further. However, BCM does not seem to correct for different hyper parameters per module, neither for badly fitted modules, which becomes a major problem when the number of modules increases.

Several other attempts have been made to divide the Kriging model in sub-models [79, 80], each solution for different domains. In [79], a *Bagging* [81] method is proposed to increase the robustness of the Kriging algorithm, rather than speeding up the algorithm's training time. In [80], a partitioning method is introduced to separate the data points into local Kriging models and combine the different models using a distance metric.

All of these approximation algorithms have their advantages and disadvantages and they are compared to our Cluster Kriging algorithms.

For the empirical study, three state of the art algorithms: *SoD*, *FITC* and *BCM* are selected to compare with the proposed approaches in this paper, as they seem to be the mostly used in their category.

5.4 Cluster Kriging

The main idea behind the proposed approach, *Cluster Kriging*, is to combine multiple Kriging models trained on each partition of data, where the partitions are obtained from clustering algorithms. Loosely speaking, if the whole data set is partitioned into clusters of similar sizes, Cluster Kriging will reduce the time complexity by a factor of k^2 resulting in $k \cdot \left(\frac{n}{k}\right)^3$ (where k is the number of clusters) if Kriging models are fitted sequentially. When exploiting k CPU processes in parallel, the time complexity will be further reduced to $\left(\frac{n}{k}\right)^3$. In practice this means that if we take k depending on n our algorithm becomes quadratic in time, and using k clusters it even reaches linear time complexity. For the output value $y^{(t)}$ at an unobserved data point $\mathbf{x}^{(t)}$, each Kriging model provides a (local) prediction for $y^{(t)}$. To obtain a global prediction, it is proposed to either combine the predictions from all the Kriging models or select the most proper Kriging model for the prediction.

There are many options for the data partitioning, e.g., K-means and Gaussian mixture models (GMM), and the Kriging model on clusters can also be combined in different manners. By varying the options in each step of the cluster Kriging, many algorithms can be generated. Four of them will be explained in the next section. In this section, the options in each step of the algorithms are introduced gradually.

5.4.1 Clustering

The first step in the Cluster Kriging methodology is the clustering of the input data $\mathcal{X}$ (and the output variables) into several smaller data sets. In general, the goal is to obtain a set $\mathcal{S}$ containing k clusters on the input data set $\mathcal{X}$.

$$\mathcal{S} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_k\}, \quad \text{where } \bigcup_{i=1}^{k} \mathcal{X}_i = \mathcal{X}. \tag{5.6}$$

In addition, the output values $\mathbf{y}$ are also grouped according to the clustering of $\mathcal{X}$: $\mathbf{y} = [\mathbf{y}_1^\top, \mathbf{y}_2^\top, \ldots, \mathbf{y}_k^\top]^\top$. The clustering can be done in many ways, with the

most simple and feasible approach being random clustering. For our framework however we introduce three more sophisticated partitioning methods that are used in the experiments later on.

Hard Clustering

The hard clustering splits the data into k smaller *disjoint* data sets:

$$\bigcap_{i=1}^{k} \mathcal{X}_i = \emptyset$$

This can be achieved by various methods, for instance the K-means algorithm (Eq. 5.7). K-means clustering minimizes the within-cluster sum of squares, that is expressed as:

$$\arg\min_{\mathrm{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in \mathcal{X}_i} ||\mathbf{x} - \boldsymbol{\mu}^{(i)}||^2, \tag{5.7}$$

where $\boldsymbol{\mu}^{(i)}$ is the centroid of cluster i and is calculated as the mean of the points in $\mathcal{X}_i$. The minimization of the within-cluster sum of squares takes only $O(nkd)$ execution time.

Fuzzy Clustering

Instead of using a hard clustering approach, a fuzzy clustering algorithm can be used to introduce slight overlap between the various smaller data sets, which might increase the final model accuracy. To incorporate fuzzy clustering, instead of directly applying cluster labels, the probabilities that a point belongs to a cluster are calculated (Eq. 5.8) and for each cluster $(n \cdot o)/k$ points with the highest membership values are assigned, where o is a user defined setting that defines the overlap. o is set between 1.0 (no overlap) and 2.0 (completely overlapping clusters).

In principle, any fuzzy clustering algorithm can be used for the partitioning. In this section the *Fuzzy C-Means* (FCM) [82] clustering algorithm and the *Gaussian Mixture Models* (GMM) [83] are used. FCM is a clustering algorithm very similar

to the well known *K-means*. The algorithm differs from K-means in that it has additional membership coefficients and a fuzzifier. The membership coefficients of a given point give the degrees that this point belongs to each cluster. These coefficients are normalized so they sum up to one. The algorithm can be fitted on a given dataset and returns the coefficients for each data point to each cluster. The number of clusters is a user defined parameter. Fuzzy C-means optimizes the objective function given in Eq. 5.8 iteratively. In each iteration, the membership coefficients of each point being in the clusters are computed using Eq. 5.9. Subsequently, the centroid of each cluster $\boldsymbol{\mu}^{(j)}$ is computed as the center of mass of all data points, taking the membership coefficients as weights. The objective of fuzzy C-means is to find a set of centroids that minimizes the following function:

$$\sum_{i=1}^{n}\sum_{j=1}^{k} w_{ij}^{m}||\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(j)}||^2, \tag{5.8}$$

where w_{ij} are the membership values (see Eq. 5.9) and m is the so-called fuzzifier (set to 2 in this chapter). The fuzzifier determines the level of cluster fuzziness as follows:

$$w_{ij}^{m} = \frac{1}{\sum_{c=1}^{k}\left(\frac{||\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(j)}||}{||\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(c)}||}\right)^{\frac{2}{m-1}}} \tag{5.9}$$

The other fuzzy clustering procedure used is the Gaussian Mixture Models. GMM are used together with the *expectation-maximization* (EM) algorithm for fitting the Gaussian models. The mixture models are fitted on the training data and later used in the weighted combination of the Kriging models by estimating cluster membership probabilities of the unseen data points. The advantage of this clustering technique is that it is fairly robust and that the number of clusters can be specified by the user. For the GMM method one could use the full covariance matrix whenever the dimensionality of the input data is small. However, when working with high-dimensional data a diagonal covariance matrix can be used instead. The time complexity of GMM depends on the underlying EM algorithm. In each iteration EM, it takes $O(nk)$ operations to re-estimate the model parameters.

Regression Tree Partitioning

The third method used is the partitioning by use of a Regression Tree [84] on the complete training set. The regression tree splits the dataset recursively at the best splitting point using the variance reduction criterion. Each leaf node of the Regression Tree represents a cluster of data points. The number of leaves (or the number of records per leave) can be set by the user. By reducing the variance in each leaf node and therefore the variance in each dataset, the Kriging models can be fitted to the local datasets much better as will be presented later on. The time complexity of using a Regression Tree for the partitioning is $O(n)$, given that the depth of the tree or the number of leaf nodes is set by the user.

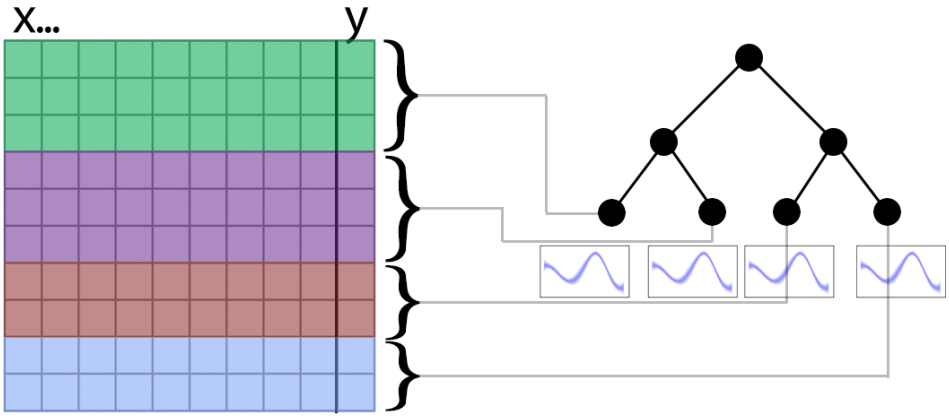


Figure 5.1: Visualisation of a Model Tree. The top node is the root and the bottom nodes are the leaves with attached models. Each record in the data (on the left) is assigned to a leaf node of the regression tree.

The partitioning done by the regression tree depends on the splitting criterion. For a faster execution of the Cluster Kriging algorithm we could choose to use a splitting criterion that splits the dataset in each node evenly, balancing the load for each of the local Kriging models attached to the leafs. From emperical experience we know that splitting using the standard variance reduction function generally results in better performing models than using such an evenly splitting criterion. This is likely due to the fact that datasets with a lower variance can be more easily fitted by a Kriging model.

5.4.2 Modeling

After partitioning the data set into several clusters, Kriging models are fitted on each of the smaller data sets. The Kriging algorithm is applied on each cluster individually, this way each model will be optimized on its own training set and will have different hyper-parameters. For simplicity we assume, in this chapter, the kernel functions used on each cluster to be the same. As for the regression tree approach, the data set, or more precisely the input space, is partitioned by the tree algorithm and, for each leaf node, a Kriging model is computed using the data belonging to this node (Fig. 5.1). A similar technique is introduced in the context of combining linear regression models [85, 86, 61]. In general, the predictive (posterior) distribution of the target variable $y^{(t)}$ on each cluster is:

$$y^{(t)} \mid \mathcal{X}_l, \mathbf{y}_l, \mathbf{x}^{(t)} \sim \mathbf{N}\left(m_l(\mathbf{x}^{(t)}), \sigma_l^2(\mathbf{x}^{(t)})\right), \quad l = 1, \ldots, k, \tag{5.10}$$

where m_l and σ_l^2 are specified again by Eq. 5.4 and 5.5 except that $\mathcal{X}, \mathbf{y}$ are replaced by $\mathcal{X}_l, \mathbf{y}_l$ here. Note that building the Kriging models can be easily parallelized, which gives an additional speedup to Cluster Kriging. Another benefit of building each model separately, is that each model has usually a much better local fit than a single global Kriging model would obtain.

5.4.3 Prediction

After training various Kriging models, unseen data points need to be predicted. For this prediction, there are several options. Depending on the partitioning method used before, the simplest way of predicting the unseen data points is by using a single local model. When the partitions are overlapping, a combination of the different local models into one global model is required.

Single Model Prediction

The most straightforward method which can be used to predict unseen data points is by using only one of the local Kriging models. This does require the partitioning used to create partitions based on locality like k-means clustering or a regression

tree. First, the partitioning method is used to predict which cluster the new data point belongs to, then the Kriging model trained using this particular cluster is used to predict the mean and variance at the new data point.

In case of the Regression Tree procedure, the targets are predicted from new unseen data points by first deciding which model needs to be used, using the Regression Tree. The target is then predicted using the specific Kriging model assigned to the leaf node (Figure 5.1). The main advantage of this method is that there is no combination of different predictions and only one of the local Kriging models needs to provide a prediction. This results in a significant speed-up for the prediction task. A possible disadvantage of this method is that you might loose the global information of the target function and that predictions close to the cluster borders may be less reliable. In Figure 5.2, a visualisation of predictions from a fitted Cluster Kriging model using regression trees is shown, marking the intersections between the different local models with black dashed lines. It can be observed that the edges of the local models are not completely matching, meaning that the predictions near the border are not as smooth as they would be in a global Kriging model. It can also be observed that the area covered by each cluster is not the same, this is due to the splitting criterion of the regression tree. While the splitting criterion could be choosen in such a way that it balances the cluster sizes, using variance reduction as the splitting criterion generally gives better fitted local models.

Optimal weighting procedure

Instead of using single model predictions, the multiple local models can be combined into one global model using various combination procedures. When the input data set is separated by hard clustering methods, the Gaussian processes built on different clusters are independent from each other. In this sense, it is possible to construct a global Gaussian process model as the superposition of Gaussian processes from all the clusters. In addition, a weighting scheme is used to model how much "trust" should be put on the prediction from each cluster. The

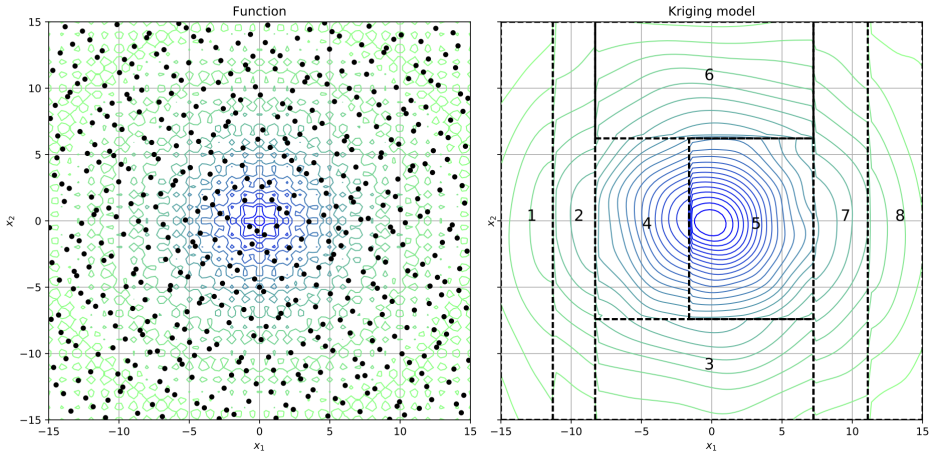


Figure 5.2: The landscape of the two dimensional Ackley function on the left, and on the right is the contours of the Model Tree Cluster Kriging mean function, with the tree partitioning visualized by dashed lines. The index of the clusters are shown in the middle of each rectangle.

weighted superposition of all Gaussian processes is [55]:

$$y^{(t)} \mid \mathcal{X}, \mathbf{y}, \mathbf{x}^{(t)} \sim \mathcal{N}\left(\sum_{l=1}^{k} w_l m_l(\mathbf{x}^{(t)}), \sum_{l=1}^{k} w_l^2 \sigma_l^2(\mathbf{x}^{(t)})\right)$$

The overall prediction and its variance depend on the weights used in the equation above. Intuitively, the optimal prediction is achieved when the variance of the estimation is minimal. To obtain such an optimal predictor, the overall Kriging variance should be optimized with respect to the weights, resulting in the following optimization task:

$$\begin{aligned} &\underset{\{w_1,\ldots,w_k\}}{\text{minimize}} && \sum_{l=1}^{k} w_l^2 \sigma_l^2(\mathbf{x}^{(t)}) \\ &\text{subject to} && \sum_{l=1}^{k} w_l = 1, \quad w_l \geq 0, \quad l = 1, \ldots, k. \end{aligned}$$

The optimal weights are obtained by solving the problem above (see [55, 54] for details):

$$w_l^* = \frac{1/\sigma_l^2(\mathbf{x}^{(t)})}{\sum_{i=1}^{k} 1/\sigma_i^2(\mathbf{x}^{(t)})}. \tag{5.11}$$

The optimal weights are then used to construct the optimal predictor, which is the inner product of the model predictions with the optimal weights.

Membership Probabilities

For the GMM and other soft clustering approaches, the membership probabilities can be used for unseen records to define the weights for the combination of predictions. For each unseen record, the membership probabilities that this record belongs to the k clusters are calculated and directly used as the weights in the weighted sum of predictions and variances given by the Kriging models:

$$w_l = \Pr(C = l \mid \mathcal{X}, \mathbf{x}^{(t)}), \quad \text{for } l = 1, \ldots, k \tag{5.12}$$

where C is the cluster indicator variable ranging from 1 to k. The rationale behind such a weighting scheme can be shown from the following derivation. In general, the goal here is to express the predictive distribution of variable $y^{(t)}$ that is the conditional density function on the whole data set $\mathcal{X}$, using the posterior densities from all clusters. By applying the total probability with respect to the cluster indicator variable C, such a density function p can be written as [54]:

$$\begin{aligned} & p(y^{(t)} \mid \mathcal{X}, \mathbf{y}, \mathbf{x}^{(t)}) \\ &= \sum_{l=1}^{k} p(y^{(t)}, C = l \mid \mathcal{X}, \mathbf{y}, \mathbf{x}^{(t)}) \\ &\approx \sum_{l=1}^{k} p(y^{(t)} \mid \mathcal{X}_l, \mathbf{y}_l, \mathbf{x}^{(t)}) \Pr(C = l \mid \mathcal{X}, \mathbf{x}^{(t)}) \end{aligned} \tag{5.13}$$

The independence assumption between Gaussian process models still holds approximately when the amount of the overlap between clusters is small. Thus, the density function $g(y^{(t)} \mid \mathcal{X}, \mathbf{y}, \mathbf{x}^{(t)})$ *approximately* equals to Eq. 5.13. The first

term inside the sum in Eq. 5.13 is the predictive density function obtained from each cluster. The second term represents the probability of data point $\mathbf{x}^{(t)}$ belonging to a cluster, which is the weight in Eq. 5.12. Consequently, the overall predictive density function is a *mixture* of predictive distributions of all the Gaussian process models on clusters. To predict $y^{(t)}$, the expectation of the conditional density function $p(y^{(t)} \mid \mathcal{X}, \mathbf{y}, \mathbf{x}^{(t)})$ is calculated:

$$\begin{aligned}
&\mathbf{E}[y^{(t)} \mid \mathcal{X}, \mathbf{y}, \mathbf{x}^{(t)}] \\
&= \sum_{l=1}^{k} \int_{-\infty}^{\infty} y^{(t)} \sum_{l=1}^{k} g(y^{(t)} \mid \mathcal{X}_l, \mathbf{y}_l, \mathbf{x}^{(t)}) \Pr(C = l \mid \mathcal{X}, \mathbf{x}^{(t)}) \, \mathrm{d}y^{(t)} \\
&= \sum_{l=1}^{k} \Pr(C = l \mid \mathcal{X}, \mathbf{x}^{(t)}) \mathbf{E}[y^t \mid \mathbf{X}_l, \mathbf{y}_l, \mathbf{x}^t] \\
&= \sum_{l=1}^{k} w_l m_l(\mathbf{x}^{(t)})
\end{aligned} \tag{5.14}$$

Note that $m_l(\mathbf{x}^t)$ is the mean function as shown in Eq. 5.10 and 5.14 suggests that the overall prediction made on the whole data set can be expressed as a convex combination of the local predictions on each cluster of data, in which the combination weights are membership probabilities of GMM or similar clustering approaches. Furthermore, the variance of the prediction (expectation) above is derived as follows:

$$\begin{aligned}
&\mathrm{Var}[y^{(t)} \mid \mathcal{X}, \mathbf{y}, \mathbf{x}^{(t)}] \\
&= \mathbf{E}[{y^{(t)}}^2 \mid \mathcal{X}, \mathbf{y}, \mathbf{x}^{(t)}] - \mathbf{E}[y^{(t)} \mid \mathcal{X}, \mathbf{y}, \mathbf{x}^{(t)}]^2 \\
&= \sum_{l=1}^{k} w_l \left(\mathrm{Var}[y^{(t)} \mid \mathcal{X}_l, \mathbf{y}_l, \mathbf{x}^{(t)}] + \mathbf{E}[y^{(t)} \mid \mathcal{X}_l, \mathbf{y}_l, \mathbf{x}^{(t)}]^2 \right) \\
&\quad - \mathbf{E}[y^{(t)} \mid \mathcal{X}, \mathbf{y}, \mathbf{x}^{(t)}]^2 \\
&= \sum_{l=1}^{k} w_l \left(\sigma_l^2(\mathbf{x}^{(t)}) + m_l^2(\mathbf{x}^{(t)}) \right) - \left(\sum_{l=1}^{k} w_l m_l(\mathbf{x}^{(t)}) \right)^2
\end{aligned} \tag{5.15}$$

Note that $\sigma_l^2(\mathbf{x}^{(t)})$ is again the Kriging variance at point $\mathbf{x}^{(t)}$ from cluster l.

5.5 Flavors of Cluster Kriging

Using the three stages and various components for each stage of the Cluster Kriging methodology, various algorithms can be implemented. In this chapter we asses four different flavors of Cluster Kriging:

Optimally Weighted Cluster Kriging (OWCK), which uses a hard (K-means) clustering technique to partition the data into k clusters. Subsequently, a Kriging model is trained on each cluster and to predict unseen data points, the predictions and variances of each model are combined using the Optimal Weights Procedure (Subsection 5.4.3).

Optimally Weighted Fuzzy Cluster Kriging (OWFCK), which uses a soft clustering technique (Fuzzy C-Means) to partition the data into k overlapping clusters and also uses the Optimal Weights Procedure combining the different predictions (Subsection 5.4.3).

Gaussian Mixture Model Cluster Kriging (GMMCK), which uses Gaussian Mixture Models to partition the data into k overlapping clusters and the trained Kriging models are weighted using the membership probabilities assigned on the unseen data by the Gaussian Mixture Model (Subsection 5.4.3).

Model Tree Cluster Kriging (MTCK), the proposed novel algorithm, uses a regression tree with a fixed amount of leaf nodes to partition the data in the objective space. A Kriging model is then trained on each partition defined by the tree's leaves. MTCK uses only one of the trained Kriging models per unseen record to predict (Subsection 5.4.3), depending on which leaf node the unseen record is assigned to.

First a decision tree regressor is constructed using the complete dataset. The tree is generated from the root node by recursively splitting the training data using the target variable and the variance reduction criterion. Once a node contains less than the minimum samples needed to split or the node contains only one record, the splitting stops and the node is called a leaf. To control the number of clusters, the user can set the maximum number of leaves or

the minimum leaf size. Next, each leaf node is assigned a unique index and each record belonging to the leaf is assigned to this index. For each leaf, a Kriging model is computed using only those records assigned to this leaf. Each Kriging model is now able to predict a particular region defined by the Regression Tree.

For the prediction of the target for unseen records, the regression tree decides which Kriging model should be used. The final predicted mean and variance is provided by this Kriging model.

5.6 Experimental Setup and Results

A broad variety of experiments is executed to compare Optimally Weighted Cluster Kriging and its Fuzzy and Model Tree variants, to a wide set of other Kriging approximation algorithms. The algorithms included in the test are; *Bayesian Committee Machines*, both with shared parameters (BCM sh.) and with individual parameters (BCM), *Subset of Data* (SoD), *Fully Independent Training Conditional* (FITC), *Optimally Weighted Cluster Kriging* (OWCK) using K-means clustering, Fuzzy Cluster Kriging using Fuzzy C-means (OWFCK), Fuzzy Cluster Kriging with Gaussian Mixture Models (GMMCK) and, finally, Model Tree Cluster Kriging (MTCK).

The above algorithms are evaluated on three different data sets from the *UCI machine learning repository* [25]:

- *Concrete Strength* [87], a data set with 1030 records, 8 attributes and one target attribute. The task is to predict the strength of concrete.
- *Combined Cycle Power Plant* (CCPP) [88], a data set of 9.568 records, 3 attributes and one target attribute. The target is the hourly electrical energy output and the task is to predict this target.
- *SARCOS* [89], a data set from *gaussianprocess.org* with a training set of 44.484 records, 21 attributes and 7 target attributes. The task is to predict the joint torques of a anthropomorphic robot arm. All 21 attributes are

used as training data but only the 1^{st} target attribute is used as target. The dataset comes with a predefined test set of 4.449 records.

In addition, 8 synthetic datasets with each 10.000 records, 20 attributes and one target attribute are used. The synthetic datasets are generated using benchmark functions from the Deap Python package [90] and are often used in optimization. The functions are *Ackley*, *Schaffer*, *Schwefel*, *Rastrigin*, *H1*, *Rosenbrock*, *Himmelblau* and *Diffpow*.

5.6.1 Hyper-Parameters

The hyper-parameters in each Kriging model are set via the Maximum Likelihood Estimation (MLE). As the constant trend μ also needs to be estimated, we use the so-called *concentrated* log likelihood for the estimation. For this numerical optimization task, we adopt a quasi-Newton method (BFGS) [91] with restarting heuristic. Each of the Kriging approximation algorithms has a hyper-parameter that can be tuned by the user to define the number of data points, clusters or inducing points, basically defining the trade-off between complexity and accuracy. For each of the algorithms a wide range of these hyper-parameters are used to see the effect and make a fair comparison between the different algorithms. The overlap for each of the Fuzzy algorithms is set to 10%, since from empirical experience we know that 10% works well. Although higher percentages (above 10%) usually increase accuracy, the increase of accuracy is not significant and costs additional training time as well. For the Model Tree variant, the number of leaves is enforced by setting a minimum number of data points per leaf and an optional maximum number of leaves. For the *Concrete Strength* dataset and all synthetic datasets: FITC is set to a range of inducing points starting from 32 and increasing in powers of 2 to 512. SoD is set to the same range as FITC but for SoD this means the number of data points. BCM, both shared and non-shared versions and all *Cluster Kriging* variants are set to a range from 2 to 32 clusters, increasing with powers of 2. For the *Combined Cycle Power Plant* dataset: FITC is set to a range of inducing points starting from 64 and increasing in powers of 2 to 1024. *SoD* is set to the a range from 256 to 4.092 data points. BCM, both shared and

non-shared versions and all *Cluster Kriging* variants are set to a range from 4 to 64 clusters.

Finally, for the *SARCOS* dataset, the range of FITC's inducing points stays the same as for the CCPP dataset, for SoD the range is from 512 to 8.184 data points, and for all cluster based algorithms and the model tree variant, the range is set from 8 to 128 clusters.

5.6.2 Quality Measurements

The quality of the experiments is estimated with the help of 5-fold cross validation, except of the *SARCOS* dataset, which uses its predefined test set. The experiments are performed in a test framework similar to the framework proposed by *Chalupka, K. et al.* [67], i.e., several quality measurements are used to evaluate the performance of each algorithm. The *Coefficient of determination* R^2 score, *Mean Standardized Log Loss* (MSLL) (see [57] Section 8.1) and the *Standardized Mean Squared Error* (SMSE) are measured for each test run. The *Mean Standardized Log Loss* is a measurement that takes both the predicted mean and the predicted variance into account. Penalizing wrong predictions that have a small predicted variance more than wrong predictions with a large variance.

$$MSLL = \left\langle \frac{1}{2} \cdot log(\pi\sigma_i + (y_i - \hat{y}_i)^2/\sigma_i) - triv \right\rangle$$

Where σ^t is the predicted variance for record $\mathbf{x}_i$ and $\hat{y}_i$ the predicted mean. With *triv* the trivial score simulating a predictor that predicts the overall mean and standard deviation:

$$triv = \frac{1}{2} \cdot log(\pi\sigma_y + (y_i - \bar{y})^2/\sigma_y)$$

For MSLL and SMSE lower scores are better, for R^2, 1.0 is the best possible score meaning a perfect fit and everything lower is worse.

Table 5.1: Average R^2 score per dataset for each algorithm

Dataset	SOD	OWCK	GMMCK	OWFCK	FITC	BCM	BCM sh.	MTCK
Concrete	0.784	0.826	0.839	0.696	0.675	-81.888	-242.459	**0.851**
CCPP	0.948	0.937	**0.968**	0.916	0.890	0.220	-24.602	**0.968**
Sarcos	0.964	0.894	0.996	0.570	0.941	-627.280	0.448	**0.999**
Ackley	0.952	0.957	0.951	0.954	0.260	0.921	-0.039	**0.981**
Schaffer	0.321	0.388	0.369	0.406	0.208	0.452	-0.050	**0.672**
Schwefel	0.990	0.973	0.977	0.947	0.006	0.969	-0.043	**0.999**
Rast	0.973	0.947	0.948	0.932	0.322	0.914	-0.043	**0.998**
H1	0.676	-0.082	0.527	-1.125	0.165	0.657	-0.046	**0.977**
Rosenbrock	0.999	0.997	0.997	0.981	0.000	0.994	-0.050	**1.000**
Himmelblau	0.997	0.995	0.995	0.981	0.291	0.994	-0.044	**1.000**
Diffpow	0.995	0.991	0.991	0.975	0.001	-0.001	-0.001	**1.000**

Table 5.2: Average MSLL score per dataset for each algorithm

Dataset	SOD	OWCK	GMMCK	OWFCK	FITC	BCM	BCM sh.	MTCK
Concrete	-0.837	-0.946	-1.100	-0.692	-0.629	18.590	68.013	**-1.140**
CCPP	-0.089	-1.438	**-1.525**	-1.109	-1.165	7.826	69.346	-1.193
Sarcos	-1.926	-1.371	-3.147	-0.302	-1.463	780.090	507.721	**-3.429**
Ackley	-1.622	-1.516	-1.517	-1.462	-0.104	7.352	13.010	**-2.012**
Schaffer	0.477	-0.073	0.081	-0.091	-0.107	16.872	11.707	**-0.514**
Schwefel	-2.554	-2.013	-2.162	-1.944	-0.002	-0.144	12.034	**-3.278**
Rast	-2.179	-1.686	-1.807	-1.642	-0.193	4.554	11.590	**-2.901**
H1	-0.766	-0.276	-0.540	-0.060	-0.059	9.018	17.393	**-1.967**
Rosenbrock	-3.479	-2.915	-3.074	-2.738	high*	0.612	18.575	**-4.054**
Himmelblau	-3.204	-2.646	-2.790	-2.553	-0.193	-1.422	12.826	**-3.739**
Diffpow	-3.020	-2.548	-2.666	-2.438	high*	high*	high*	**-3.744**

5.6.3 Results

The results of experiments on the real world data sets *Concrete Strength*, *CCPP* and *SARCOS* are shown in Figure 5.3 and the results on the Synthetic data sets

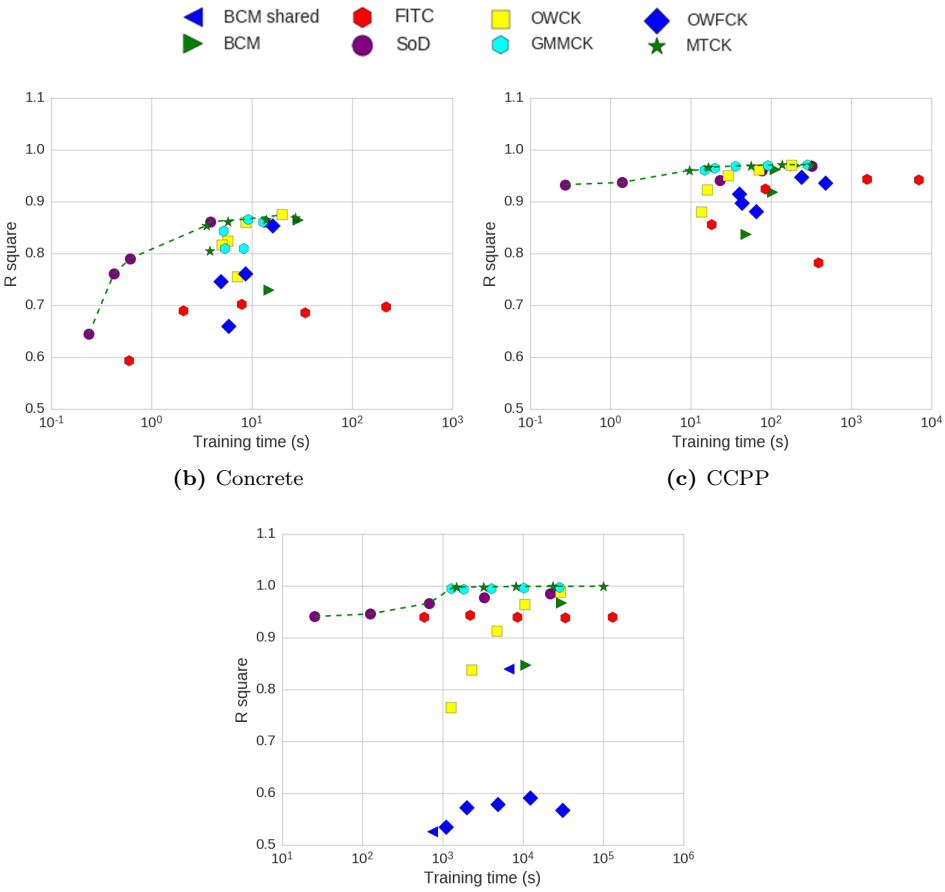
Table 5.3: Average SMSE score per dataset for each algorithm

Dataset	SOD	OWCK	GMM-CK	FCM-CK	FITC	BCM	BCM sh.	MTCK
Concrete	0.216	0.174	0.161	0.304	0.325	82.888	243.459	**0.149**
CCPP	0.052	0.063	**0.032**	0.084	0.110	0.780	25.602	**0.032**
Sarcos	0.036	0.106	0.004	0.430	0.059	628.280	0.552	**0.001**
Ackley	0.048	0.043	0.049	0.046	0.740	0.079	1.039	**0.019**
Schaffer	0.679	0.612	0.631	0.594	0.792	0.548	1.050	**0.328**
Schwefel	0.010	0.027	0.023	0.053	0.994	0.031	1.043	**0.001**
Rast	0.027	0.053	0.052	0.068	0.678	0.086	1.043	**0.002**
H1	0.324	1.082	0.473	2.125	0.835	0.343	1.046	**0.023**
Rosenbrock	0.001	0.003	0.003	0.019	1.000	0.006	1.050	**0.000**
Himmelblau	0.003	0.005	0.005	0.019	0.709	0.006	1.044	**0.000**
Diffpow	0.005	0.009	0.009	0.025	0.999	1.001	1.001	**0.000**

are shown in Figures 5.4 and 5.5. The results are shown with both objectives, time and accuracy (x and y axis respectively) in mind to show the trade-off and to show that some algorithms are performing better in both objectives. The R^2 scores of each dataset per algorithm, averaged over all folds, are shown in Table 5.1. The MSLL scores are provided in Table 5.2 and the SMSE scores in Table 5.3. The best results for each dataset are shown in bold face.

5.6.4 Parameter Setting Recommendations

To use the Cluster Kriging algorithms, the minimum cluster size or the number of clusters has to be set as a user defined parameter. It is recommended to set this parameter in such a way that each individual cluster contains between 100 and 1000 records. 1000 records is still computationally tractable by Kriging in terms of execution time and 100 records is in most cases still doable in terms of fitting the Kriging model. Selecting smaller cluster sizes is likely to result in poorly fitted models and selecting cluster sizes larger than 1000 will in most cases not increase accuracy but will only increase execution time. These recommendations are purely based on empirical observations and depend highly on the dataset one is working

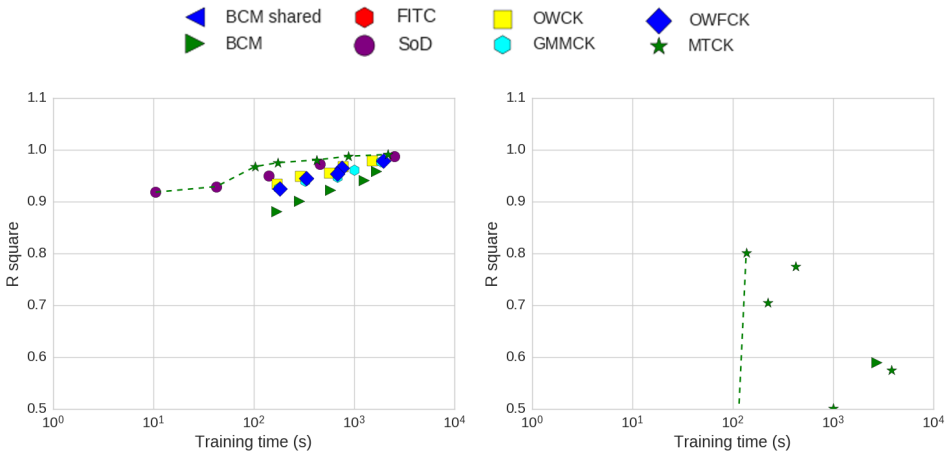


(b) Concrete **(c)** CCPP

(d) Sarcos

Figure 5.3: Quality measurements of each algorithm with the hyper-parameters increasing in sample sizes for FITC and SoD, and decreasing in cluster sizes for the cluster based algorithms as explained in Subsection 5.6.1. The results are shown for the Concrete, CCPP and Sarcos datasets. The training time is given on the x axis and the R^2 score on the y axis. The dashed green line indicates the non dominated set.

with. For MTCK smaller cluster sizes are usually still fine because of the low variance in the records per leaf due to the splitting criterion of the Regression Tree.



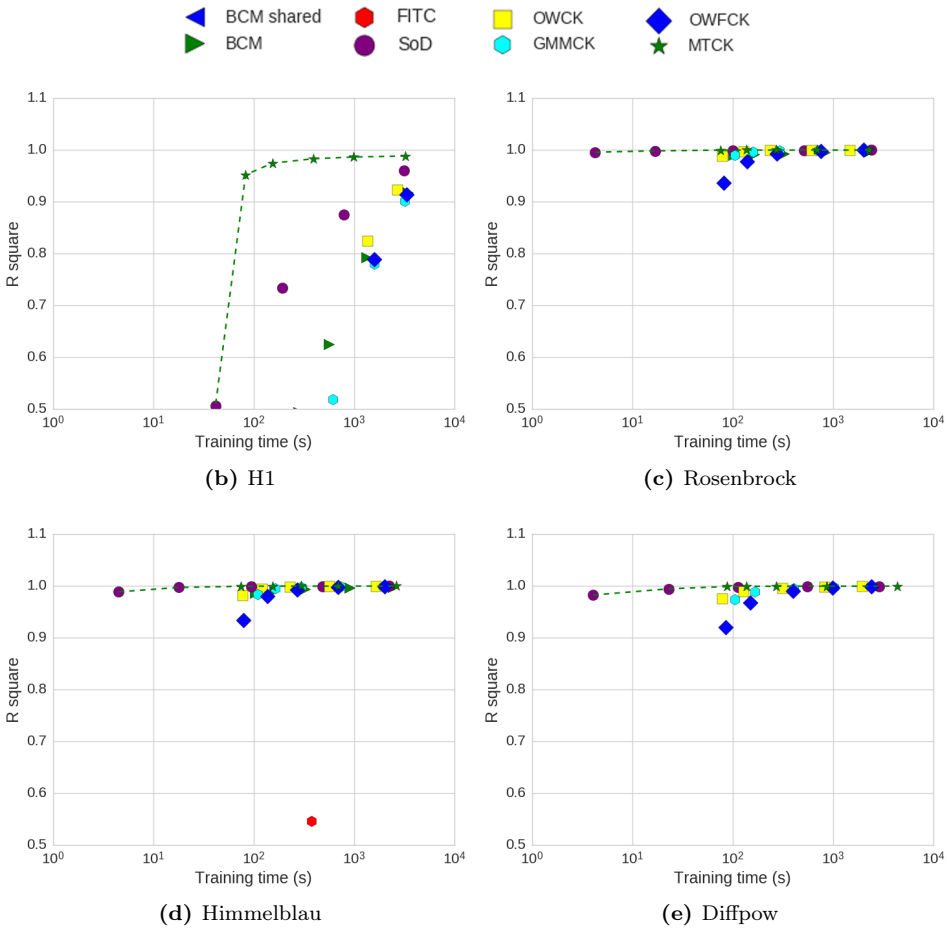
(b) Ackley **(c)** Schaffer

(d) Schwefel **(e)** Rastrigin

Figure 5.4: Quality measurements of each algorithm with the hyper parameters increasing in sample sizes for FITC and SoD, and decreasing in cluster sizes for the cluster based algorithms on the first half of synthetic datasets. The training time on the x axis and the R^2 score on the y axis. The green line indicates the non dominated set.

5.7 Efficient Global Optimization

In many real-world optimization problems, such as optimizing the manufacturing of car body parts or the production of steel, function evaluations are costly, either in time or money. *Efficient Global Optimization* (EGO) [70] is a procedure



(b) H1 **(c)** Rosenbrock

(d) Himmelblau **(e)** Diffpow

Figure 5.5: Quality measurements of each algorithm with the hyper parameters increasing in sample sizes for FITC and SoD, and decreasing in cluster sizes for the cluster based algorithms on the second half of the synthetic datasets. The training time on the x axis and the R^2 score on the y axis. The green line indicates the non dominated set.

designed to use a very low number of function evaluations while optimizing a specific function. The procedure uses a surrogate model to approximate the response surface of the real function. The surrogate model is fitted using an initial space filling *Design of Experiments* (DOE) [92]. Once the surrogate model is fitted on this data, optimization on the surrogate model's response surface can be performed

to find good candidate solutions for the black-box function to be optimized. This step does not require any additional expensive function evaluations since it uses the surrogate model. For the selection of these candidate points, EGO uses an infill-criterion, which is meant to provide a nice balance between exploration and exploitation. The newly found candidate solution is then evaluated against the black-box function and added to the data set and used to re-fit the surrogate model. This procedure is repeated untill the convergence criteria are met.

The Efficient Global Optimization [70] or Bayesian optimization [93, 94] is a sequential model-based global optimization algorithm that is built on stochastic models over the unknown objective function. The Kriging modeling technique [95] is originally proposed as the underlying model in EGO.

5.7.1 The Efficient Global Optimization Algorithm

EGO [70] is proposed to optimize expensive objective functions by sequentially choosing new candidate solutions from an underlying Kriging model. The candidate solutions are obtained by maximizing the so-called *acquisition function* or infill-criterion. Acquisition functions usually take the mean and variance of the posterior process (Eq. 5.10) into account, in order to balance the exploration and exploitation of the global search. Adding the newly obtained data points into the underlying Kriging model, its posterior process is modified and the acquisition function is updated accordingly. In this manner, a sequence of new solutions are generated iteratively. This algorithm is summarized in Algorithm 5.1. Many acquisition functions have been proposed and investigated [96]. The most popular ones are: Lower Bound (LB) [97], the Probability of Improvement (PI) [98, 99] and Expected Improvement (EI) [70]. In this chapter, we focus only on the expected improvement, that is defined as follows, in terms of minimization:

$$\begin{aligned} \mathrm{EI}(\mathbf{x}) &= \mathbf{E}[\max\{0, \min(\mathbf{y}) - y(\mathbf{x})\} \mid \mathbf{y}] \\ &= (\min(\mathbf{y}) - m(\mathbf{x}))\Phi\left(\frac{\min(\mathbf{y}) - m(\mathbf{x})}{s(\mathbf{x})}\right) \\ &+ s(\mathbf{x})\phi\left(\frac{\min(\mathbf{y}) - m(\mathbf{x})}{s(\mathbf{x})}\right) \end{aligned} \tag{5.16}$$

Algorithm 5.1 Efficient Global Optimization

1 Generate the initial data set $\mathcal{X}, \mathbf{y}$
2 Fit the Kriging model hyper-parameters on the initial data set $\mathcal{X}, \mathbf{y}$.
3 **while** the stop criteria are not fulfilled **do**
4 Find global optimum of the infill criterion:

$$\mathbf{x}^* = argmax_{\mathbf{x}} \, \mathrm{EI}(\mathbf{x})$$

5 Evaluate $\mathbf{x}^*$: $y^* = y(\mathbf{x}^*)$ and append $\mathbf{x}^*, y^*$ to $\mathcal{X}, \mathbf{y}$.
6 Re-estimate the Kriging model hyper parameters
7 **end while**

where $\Phi(\cdot), \phi(\cdot)$ denote the cumulative distribution function and the probability density function of the standard normal distribution, respectively. It takes into account the quantity of the expected improvement and also rewards a higher variance. In addition, the gradient of the expected improvement is given in Equation 5.17, as it is required by the quasi-Newton optimization procedure, that is used in the next subsections.

$$\begin{aligned} \nabla \, \mathrm{EI}(\mathbf{x}) &= \phi(u) \nabla s(\mathbf{x}) - \Phi(u) \nabla m(\mathbf{x}) \\ u &= \frac{\min(\mathbf{y}) - m(\mathbf{x})}{s(\mathbf{x})} \end{aligned} \tag{5.17}$$

When applying the EGO algorithm to a large initial data set (e.g., in the experiment design), the CPU time spent on the hyper-parameter re-estimation becomes computationally infeasible. To relax this issue, it is proposed to use time complexity reduction techniques that have been developed for the Kriging model.

5.7.2 Cluster Kriging-based EGO

It is proposed to exploit the Cluster Kriging variants in the EGO algorithm, for time complexity reduction. Although various complexity reduction (or approximation) methods exist for Kriging, we state that Cluster Kriging is more suitable for the EGO algorithm for the following reasons.

Firstly, the Kriging models on each cluster can be executed in parallel, which yields an additional linear speedup in practice. Secondly, after a new candidate solution is found through the acquisition function, the hyper-parameters of Kriging needs to be re-estimated. Taking the cluster information into account, it is proposed to only re-estimate the Kriging models on the clusters that this new solution belongs to. This operation results in another linear speedup in the hyper-parameter re-estimation procedure, as in the best scenario, only one Kriging model is subject to re-fitting. Thirdly, the acquisition function, e.g., the expected improvement is still well-defined on Cluster Kriging because either the posterior process (Eq. 5.10) or at least the mean and variance function (Eq. 5.14) can be derived. The algorithm is presented in Algorithm 5.2.

In the algorithm, the initial fitting procedure can be parallelized (line 2). Usually, the cluster (and the Kriging model on it) that the new solution belongs to is updated (line 13-16). A counter c is incremented every time when a new candidate solution is generated (line 5). If the c value, that is the recently appended data points, are more than 10% of the initial data set, the clustering is performed again to keep the size of each cluster balanced and capture the information contained in the newly added points.

5.7.3 Experiments

Several experiments are conducted to show both the empirical time complexity and convergence rate of the proposed Cluster Kriging based EGO, including all the variants of Cluster Kriging discussed earlier in this chapter. The performance of the proposed algorithm is compared to the original EGO that uses *Ordinary Kriging* (OK). For our experiments, the benchmark functions chosen are *Ackley*, *Rastrigin* and *Schaffer*. These functions are chosen because they are used often in optimization experiments, are highly multi modal, and are of a relatively high complexity.

Experiment 5.1 The algorithms compared are: EGO with Ordinary Kriging (OK), Tree-based local Kriging models (MTCK), Superposition of Kriging models (OWCK) and the mixture of Kriging models (GMMCK). Each of the Cluster Kriging variants uses 5 clusters. Both execution time and convergence rate are

Algorithm 5.2 Cluster Kriging based Efficient Global Optimization (CK-EGO)

Input: Data set $\mathcal{X}, \mathbf{y}$ obtained on a black-box function f. The number of clusters q. The clustering method is chosen from K-means, GMM or regression trees by the user.

Initial Clustering: $\{\mathcal{X}_i, \mathbf{y}_i\}_{i=1}^{q} \leftarrow \mathcal{X}, \mathbf{y}$
Create the Kriging model for each cluster:

$$y \mid \mathcal{X}_i, \mathbf{y}_i \sim \mathbf{\mathcal{N}}\left(m_i(\mathbf{x}), s_i^2(\mathbf{x})\right), \quad i = 1, \ldots, q$$

$c \leftarrow 0$
while the stop criteria are not fulfilled **do**
  $\mathbf{x}^* = argmax_{\mathbf{x}}\, \mathrm{EI}(\mathbf{x})$
  Evaluation: $y^* = f(\mathbf{x}^*)$
  $c \leftarrow c + 1$
  if $c > 10\%$ the number of data points in $\mathcal{X}$ **then**
    Merge the data set: $\mathcal{X}, \mathbf{y} \leftarrow \{\mathcal{X}_i, \mathbf{y}_i\}_{i=1}^{q}$
    Clustering the data set $\mathcal{X}, \mathbf{y}$ and re-create the Kriging models for each cluster.
    $c \leftarrow 0$
  else
    for every cluster i that $\mathbf{x}^*$ belongs to **do**
      Append $\mathbf{x}^*, y^*$ to $\mathcal{X}_i, \mathbf{y}_i$.
      Re-estimate the hyper-parameter for the Kriging model on cluster i.
    end for
  end if
end while
return $\mathbf{x}^*$

being measured with a fixed set of EGO iterations and optimization budget. The convergence is measured by taking the absolute error between the real optimum of the benchmark functions and the found optimum for each iteration of EGO. Each EGO run performs 10 iterations for the three benchmark functions in two dimensions. Three different initial sample sizes are used to train the surrogate models, $500, 1000$ and 5000 points in order to illustrate the growth of CPU time required per algorithm, when the size of the data available increases. For each different experimental setup, the average time and distance to the optimum is recorded over 20 runs with different random seeds.

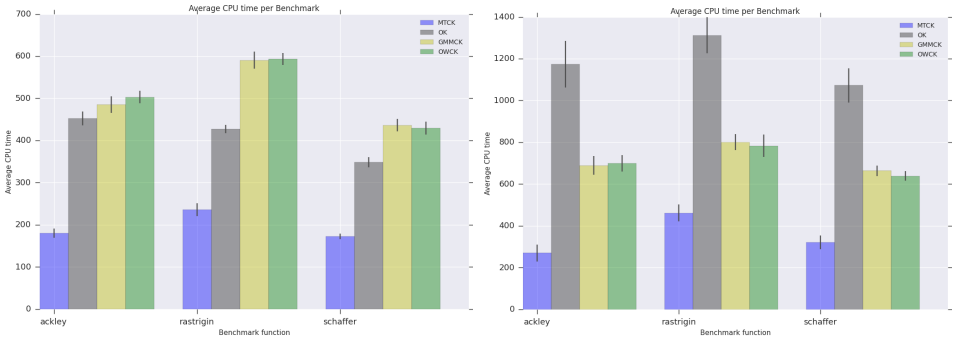
Experiment 5.2 The algorithms, OK, MTCK and OWCK are compared in five dimensions on the benchmark functions Ackley and Rastrigin also varying the algorithm that maximizes the expected improvement. CMA-ES and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm are compared.

Results From Figure 5.7 it can be observed that the Cluster Kriging based EGO variants perform very similar to OK, depending on the target function, a specific variant even outperforms Ordinary Kriging. Due to the relatively large variance in the results it is difficult to judge which algorithm performs better. However, from the CPU time in Figure 5.6 it can be observed that Cluster Kriging and in particular MTCK takes only a fragment of the time that Ordinary Kriging requires. Using a sample size of 500 points this difference is mainly due to the re-fitting of only one local model at a time. This can be seen by comparing MTCK with GMMCK and OWCK, since all three cluster Kriging variants use the same number of local models and only MTCK uses an adaptive local model strategy. When the number of points increases to 1.000 and even 5.000, the difference between the three cluster Kriging variants decreases but the difference with Ordinary Kriging becomes enormous. This shows that using EGO with Ordinary Kriging quickly becomes infeasible when the number of data points grow.

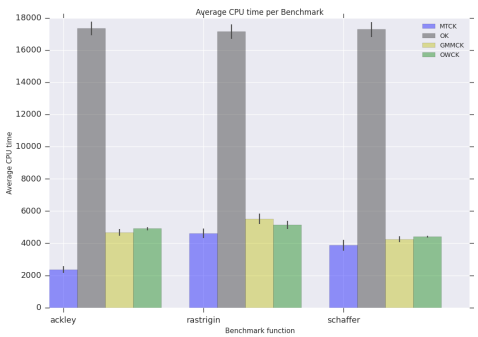
From Figure 5.8 it can be observed that also in higher dimensions Cluster Kriging does not under-perform Ordinary Kriging. In addition, it can be observed that using different optimization strategies for the expected improvement affects the convergence rate. However, the best optimization strategy clearly depends on the target function.

5.8 Conclusions

A novel Kriging approximation methodology, Cluster Kriging, is proposed, using a combination of smaller Kriging models trained on partitions of the data set. Four different algorithms using this methodology are proposed and explained in detail and a broad comparison between the novel algorithms and other state of the art Kriging approximation algorithms is done. The results of the experiments (as given in Section 5.6) clearly show that for each data set, the *Gaussian Mixture Models Cluster Kriging* (GMM CK) and the Model Tree Cluster Kriging (MTCK)

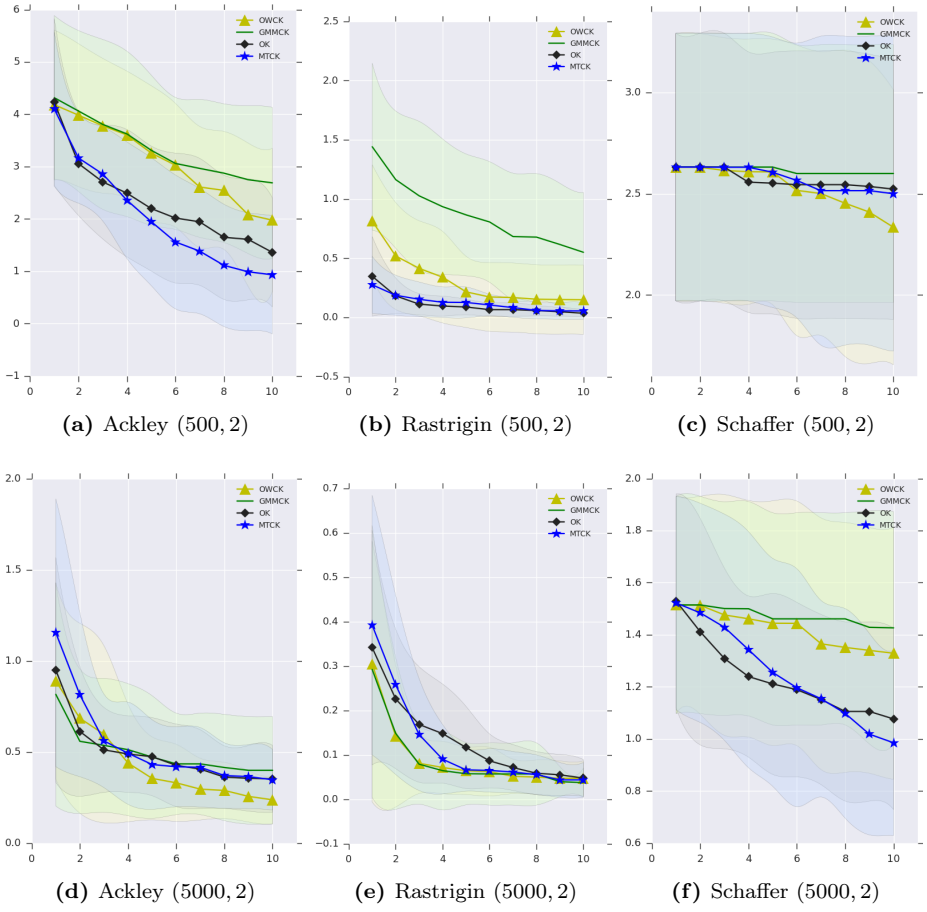


(a) CPU time $(500, 2)$ **(b)** CPU time $(1000, 2)$



(c) CPU time $(5000, 2)$

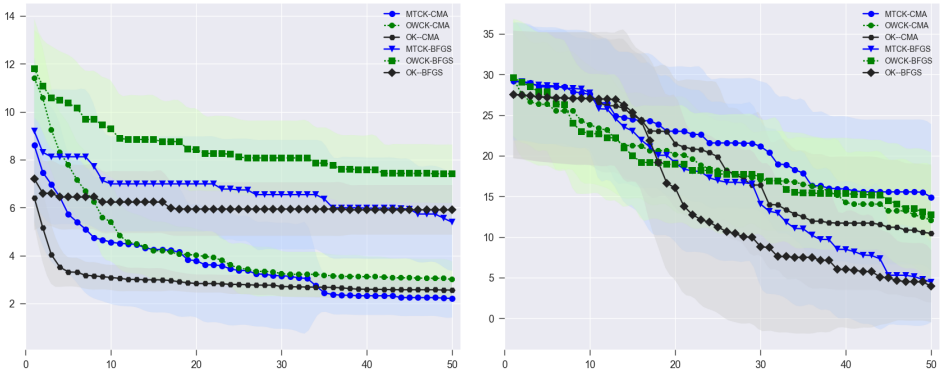
Figure 5.6: Average CPU time (in sec.) per benchmark function for varying sample sizes $(n_{samples}, d_{dimensions})$. In blue the MTCK algorithm, OK is denoted in grey, GMMCK in yellow and OWCK in green.



(a) Ackley $(500, 2)$ **(b)** Rastrigin $(500, 2)$ **(c)** Schaffer $(500, 2)$

(d) Ackley $(5000, 2)$ **(e)** Rastrigin $(5000, 2)$ **(f)** Schaffer $(5000, 2)$

Figure 5.7: Average convergence of the absolute error of three benchmark functions in two dimensions, with varying training sample sizes n and 10 iterations of EGO. Shown is the average over 20 runs (lines) and one standard deviation (shaded areas).



(a) Ackley $(500, 5)$ **(b)** Rastrigin $(500, 5)$

Figure 5.8: Average convergence of the absolute error of two benchmark functions in five dimensions using different optimization algorithms. 500 training samples and 50 iterations of EGO are used. Shown is the average over 20 runs (lines) and one standard deviation (shaded areas).

outperform the other algorithms in all measurements. It can also be observed that the *Bayesian Committee Machine* algorithms, both with shared parameters and with individual parameters, are very unstable when the number of clusters is above 8. This is most likely due to poor recombination of models with different hyper-parameters and the chance of poor fitting of one of the clusters. In terms of training time, *Subset of Data* is much faster than any of the other algorithms, though it pays for this complexity reduction by a decrease in accuracy. Both for SoD and FITC, the training time increases faster than the training time of the cluster based algorithms. It is shown that the membership probabilities of the Gaussian Mixture Model can be used as weights in the combination of the various Kriging models' predictions. It is also shown that a Model Tree of Kriging models works very well in high-dimensional problems and requires less prediction time due to the fact that only one Kriging model per unseen data point is used for prediction.

It is shown that Cluster Kriging can be applied in the EGO algorithm for complexity reduction. Three variants of Cluster Kriging are validated in combination with EGO on some test functions. Based on the empirical results that are shown

in Section 5.7, it can be concluded that EGO using Cluster Kriging is much faster in terms of time complexity compared to the traditional EGO that employs an Ordinary Kriging model. Moreover, each of the Cluster Kriging variants perform very well compared to EGO using Ordinary Kriging in terms of convergence speed. From the results shown in Subsection 5.7.2, it can be inferred that the MTCK model fits the objective function well due to the reason that it captures local information much better than Ordinary Kriging.

6

Arbitrary Model Efficient Global Optimization

The Kriging model, as stated in Chapter 5, is heavily exploited by the *Efficient Global Optimization* framework, a popular optimization algorithm for optimizing expensive functions, such as the production of high quality steel in the PROMI-MOOC project.

Applying Efficient Global Optimization with Cluster Kriging is one solution in solving the time and space complexity bottleneck of standard Kriging while using the Global Optimization algorithm. However, many data sets do not obey the assumptions that the Kriging model is based on. Other machine learning techniques such as Artificial Neural Networks, Random Forests and Support Vector Machines might perform much better as surrogate models. Unfortunately, these techniques do not provide a prediction variance out of the box.

In this chapter a novel uncertainty measure is proposed to allow the use of these different data driven models in the *Efficient Global Optimization* framework. With the help of this measure, global optimization with complex deep neural networks, as well as other machine learning techniques such as Random Forests, as the surrogate model, becomes feasible.

In this chapter the challenges of applying global optimization on arbitrary machine learning models is discussed. In order to use the models in the model-based optimization procedure of the framework (Section 2.3), an uncertainty measure, the k-NN uncertainty is presented. It is shown that the k-NN uncertainty allows for combinations of EGO and models that outperform the classical EGO using Kriging.
This chapter is primarily based on the publication [100].

6.1 Background

As discussed in Chapter 2, the main objective of the industrial partner Tata Steel, is to classify surface defects on incoming steel slabs and optimize process parameters to reduce the number of defects occurring. To achieve this objective within the framework presented in Section 2.3, global optimization algorithms like EGO have to be applied using a large scale of different predictive models as surrogate model. Using different surrogate models can significantly improve the accuracy of the final predictive model and therefore greatly increase the performance of the EGO algorithm.

However, there is one major issue, the infill-criteria of EGO requires a prediction uncertainty or prediction variance, which most statistical models do not provide out of the box. To overcome this issue, a novel uncertainty measure is introduced. This measure allows us to apply EGO in combination with any predictive model such as deep neural networks that are capable of predicting surface defects on the Tata Steel data set.

6.2 kNN Uncertainty Measure for EGO

For most regression models, their overall accuracy can be estimated with help of various error measures. However, in some applications it is important to provide not only point predictions, but also to estimate the "uncertainty" of the prediction, e.g., in terms of confidence intervals, variances, or interquartile ranges. There are very few statistical modeling techniques able to achieve this. For instance, the Kriging method is equipped with a theoretical mean squared error. In this section we address this problem by introducing a heuristic method to estimate the uncertainty of the prediction, based on the error information from the k-nearest neighbors. This heuristic, called the *k-NN uncertainty measure*, is computationally much cheaper than other approaches (e.g., bootstrapping) and can be applied regardless of the underlying regression model. To validate and demonstrate the usefulness of the proposed heuristic, it is combined with various models

and plugged into the well-known EGO algorithm. This allows for the usage of regression models other than the Kriging model.

Such a measure of the prediction uncertainty should aim at the following objectives: 1) It should operate independently of the modeling assumptions. 2) It should be exploitable by the Efficient Global Optimization algorithm, making it possible to use any regression model in the EGO framework.

In the nonparametric settings, when estimating the mean squared error of the predictor, the available information are the data set $\mathcal{D} \coloneqq \{\mathcal{X}, \mathbf{y}\}$ and the prediction $\hat{f}(\mathbf{x})$ at $\mathbf{x}$. Intuitively, this empirical uncertainty measure should be zero at correctly predicted known observations and increase for the data points that are far from the observations. Given these preferred properties, a distance-weighted measure $\widehat{U}_{k\text{-NN}}$ is proposed as follows:

$$\widehat{U}_{k\text{-NN}} = \underbrace{\frac{\sum\limits_{i \in N(\mathbf{x})} w_i^k \left| \hat{f}(\mathbf{x}) - y_i \right|}{\sum\limits_{i \in N(\mathbf{x})} w_i^k}}_{\text{empirical prediction error}} + \underbrace{\frac{\min\limits_{i \in N(\mathbf{x})} d(\mathbf{x}_i, \mathbf{x})}{\max\limits_{\mathbf{x}_i, \mathbf{x}_j \in \mathcal{X}} d(\mathbf{x}_i, \mathbf{x}_j)} \widehat{\sigma}}_{\text{variability of the observation}} \quad . \tag{6.1}$$

$$w_i = 1 - \frac{d(\mathbf{x}_i, \mathbf{x})}{\sum\limits_{i \in N(\mathbf{x})} d(\mathbf{x}_i, \mathbf{x})}, \quad \widehat{\sigma} = \sqrt{\mathrm{Var}\left[\{y_i\}_{i \in N(\mathbf{x})} \cup \{\hat{f}(\mathbf{x})\} \right]}.$$

Note that $N(\mathbf{x})$ collects the indices of k-nearest neighbors to $\mathbf{x}$ and $d(\cdot, \cdot)$ denotes the Euclidean distance metric. $\widehat{\sigma}$ is computed as the standard deviation of the observations in the neighborhood with the prediction $\hat{f}(\mathbf{x})$.

The proposed uncertainty quantification consists of two components: *1) the empirical error of the prediction and 2) the variability of the observed outputs y.* Intuitively, less empirical prediction error leads to higher certainty of the prediction. Moreover, when comparing two different regression tasks, a large variability of the observations y could contribute to the high uncertainty of the prediction, even if the predictor $\hat{f}$ were making the same empirical error on both tasks. The empirical error is computed from the difference between the prediction $\hat{f}(\mathbf{x})$ and the observations at the k-nearest neighbors. Such differences are linearly scaled

where the weights are inversely proportional to Euclidean distances to the neighbors. This heuristic is based on the intuition that the closer neighbors have more influence than neighbors that are further away. To quickly diminish the effect of far-away neighbors, the exponent k (the number of neighbors) is applied on the weights. The variability of the observation is estimated by calculating the standard deviation of the observations at the nearest neighbors and the predicted point. The resulting value is then rescaled by the distance to the nearest neighbor. Using the distances to scale the heuristic error prediction, we make sure that the uncertainty goes to zero at correctly predicted known points and that it increases when predicting points further away from the known observations.

A good number of neighbors is depending on the number of known points and the data dimensionality. For most of the experiments k is set to 20, more neighbors will provide a more smooth but also slightly more pessimistic prediction error while less neighbors make the expected prediction error more optimistic and less smooth.

To illustrate the behaviour of k-NN uncertainty, a 1-D function $f(x) = x\sin(x)$ is used in Figure 6.1. Note that k-NN uncertainty (green area) progresses very similarly to the Kriging uncertainty quantification (blue area). It can also be observed that at the known observations the prediction error given by the k-NN uncertainty algorithm is exactly the error between the prediction and the known observation. Note that the SVR model is badly fitted, and different hyper-parameters would result in a much better fit. This is on purpose to illustrate how the k-NN uncertainty would look like using less fitted models. Lastly, it can be observed that when the to be predicted point is far away from the known data points the uncertainty of the prediction increases.

6.3 Experimental Setup

Two different experimental setups are used to demonstrate the properties and effectiveness of k-NN uncertainty in Efficient Global Optimization. First, we validate k-NN uncertainty by visual inspection of plotted two and five dimensional benchmark functions that are often used in the field of optimization (in this case

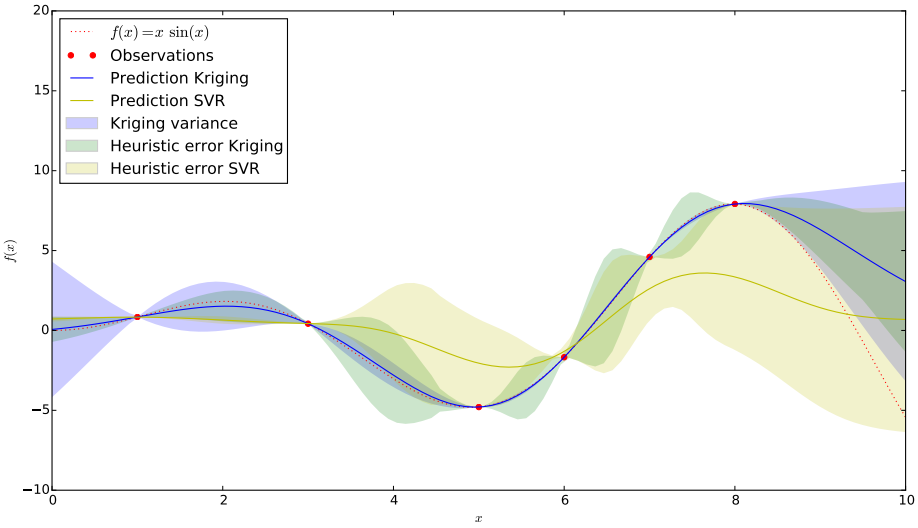
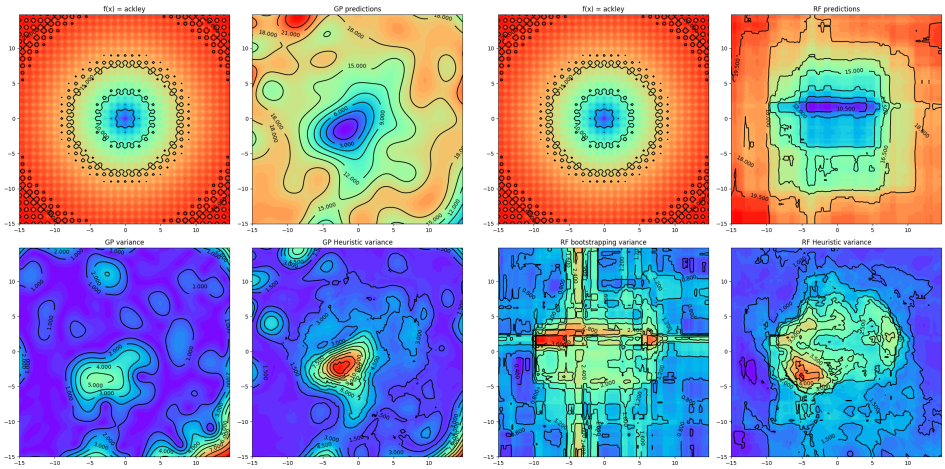


Figure 6.1: Best viewed in color. Visualization of k-NN uncertainty heuristic. The dotted red line is the real function $f(x) = x \sin x$, the red dots are the observed points. The blue line is the predicted mean of the Kriging model with the shaded blue area showing the standard Kriging variance and the shaded green area is k-NN uncertainty on the same Kriging model. The yellow line shows the predictions of a Support Vector Regression (SVR) model with default hyper-parameters ($C = 1$, RBF kernel, $\epsilon = 0.1$) with the shaded yellow area denoting k-NN uncertainty using the SVR model. The number of neighbors for k-NN uncertainty is set to 4 in this case.

the *Ackley* and *Schaffer* function). In Figure 6.2 it can be observed that k-NN uncertainty is quite similar to the Kriging variance as shown in the lower subplots of Figure 6.2 a. k-NN uncertainty is a bit less optimistic than the Kriging variance but shows roughly the same areas with higher variance. When looking at the Random Forest bootstrapping variance and k-NN uncertainty it can be observed that the bootstrapping variance is very blocky, due to the Random Forest model assumptions. k-NN uncertainty however does not use the individual tree predictions of the Random Forest model and because of the interpolation effect using the distances to the known observations, it creates a much more smooth surface. In Figure 6.3 the models are trained on samples of the *Schaffer* function in the space of -50 to 50 for both dimensions while tested on the complete range of -100

to 100. It can be observed that the k-NN uncertainty gradually increases when moving away from the known observations, while the Kriging variance almost immediately explodes to a flat high value.

When looking at the same benchmark function but now in five dimensions, we can plot a one-dimensional slice of the function (using the first dimension) to show the local behaviour of k-NN uncertainty versus the Kriging variance in Figure 6.4. Here it can be observed that the Kriging variance is over-optimistic and actually wrong, while k-NN uncertainty is much more pessimistic and actually captures very nicely the shape of the underlying function.

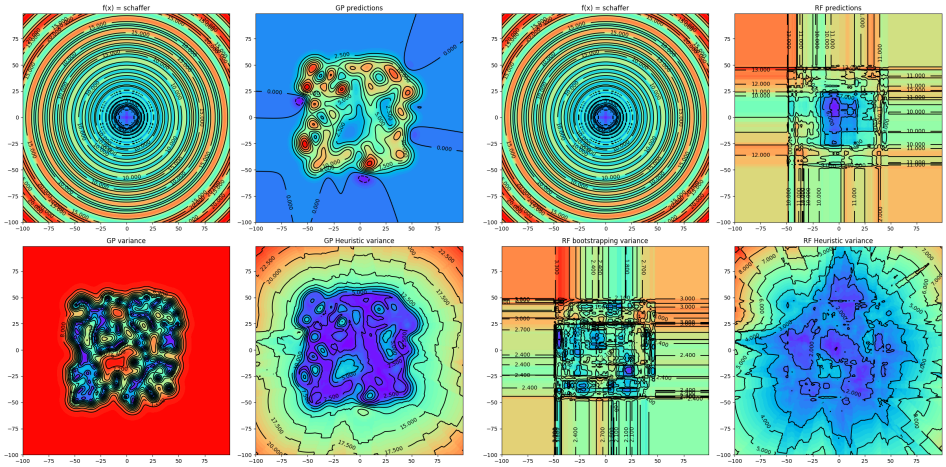


(a) Kriging variance versus k-NN uncertainty

(b) Random Forest bootstrapping variance versus k-NN uncertainty

Figure 6.2: a) Upper-left plot is the *Ackley* function in two dimensions, upper-right shows the Kriging prediction of this function using 100 data points for training. Lower-left plot shows the Kriging variance and bottom-right shows k-NN uncertainty using the same Kriging model. b) Upper-left plot is the same as a), upper-right shows a Random Forest predictor with 50 trees using 100 data points for training. Lower-left plot shows the variance given by the tree regressors of the Random Forest and bottom-right shows k-NN uncertainty using the same Random Forest model. The number of nearest neighbors for k-NN uncertainty is set to 20.

The second experiment is more quantitative as we compare the performance of k-NN uncertainty in the setting of Efficient Global Optimization. We compare



(a) Kriging variance versus k-NN uncertainty

(b) Random Forest bootstrapping variance versus k-NN uncertainty

Figure 6.3: Same as in Figure 6.2 but now using the *Schaffer* benchmark function.

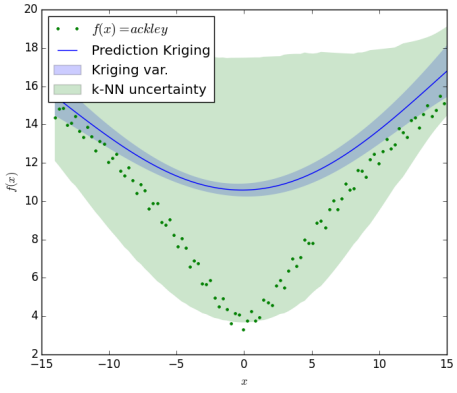


Figure 6.4: The green dots are unseen observations of the *Ackley* function in five dimensions (slice with the last four dimensions set at zero), the blue line is the predicted mean of a Kriging model, the blue shaded area is the Kriging variance and the green shaded area k-NN uncertainty with 20 nearest neighbors using the same Kriging model.

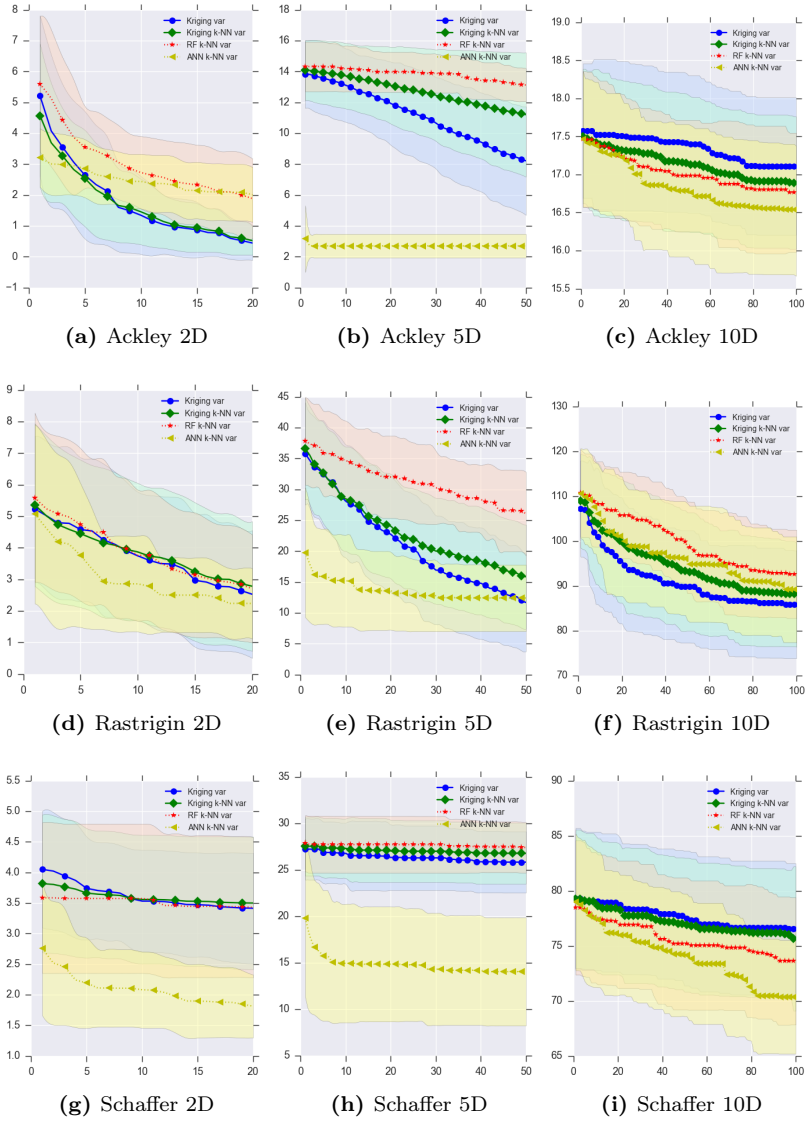
the convergence speed of the original EGO with Kriging, EGO with Kriging using k-NN uncertainty instead of the Kriging variance, EGO with a Random Forest model using k-NN uncertainty as the prediction variance and finally EGO with a multi-layer perceptron using k-NN uncertainty (using two hidden layers of size 100 and 50 nodes respectively).

The experiment is carried out using three different benchmark functions *Ackley*, *Rastrigin* and *Schaffer* with implementations from the DEAP [90] python package. For each function, the experiments are repeated using 100 initial samples using a Latin hypercube sampling strategy, and in 2, 5 and 10 dimensions (d). The EGO algorithm is run for $10 \cdot d$ evaluations and each experiment is repeated 40 times. The number of nearest neighbors for k-NN uncertainty is set to 20.

From Figure 6.5 we can observe that in most cases the convergence is very similar for all four EGO setups. In the two dimensional cases the Random Forest setup seems to be slightly worse performing than the Kriging setups, on the other hand, in the ten dimensional cases the Random Forest setup seems to outperform the Kriging setups. For the Kriging models, the original Kriging variance seems to perform slightly better than the heuristic k-NN uncertainty, however in most cases this is only marginal. Interesting is to note the performance of the neural network using k-NN uncertainty, which performs very well and even outperforms the standard EGO procedure with Kriging in the five dimensional cases. Further investigation showed us that the neural network fits the underlying global trend of the function much more accurate than the Kriging or Random Forest model, allowing the EGO procedure to quickly converge to the global optimum.

6.4 Conclusion

An uncertainty quantification measure, the k-NN uncertainty measure is proposed. The proposed heuristic works independently of the modeling assumptions and can therefore be used in combination with any regression model. It is shown that the heuristic function obeys the preferred properties: 1) it ensures exactitude; on known observations a correct prediction gives zero prediction variance. 2) it



(a) Ackley 2D **(b)** Ackley 5D **(c)** Ackley 10D

(d) Rastrigin 2D **(e)** Rastrigin 5D **(f)** Rastrigin 10D

(g) Schaffer 2D **(h)** Schaffer 5D **(i)** Schaffer 10D

Figure 6.5: Convergence on the three benchmark functions. Displaying, as blue dots, the original EGO procedure with Kriging variance, the red stars show the convergence of EGO with a Random Forest model using k-NN uncertainty, the green diamonds illustrate the convergence of EGO with Kriging using k-NN uncertainty and the yellow triangles illustrate the convergence of EGO with a multi-layer perceptron using k-NN uncertainty. The shaded areas show the 95% confidence interval over 40 runs.

increases with the dispersion of the known observations. 3) It is exactly the prediction error when predicting known data points. The behaviour of the k-NN uncertainty is verified by plotting the surface of several predictors on two benchmark functions and by running a wide set of experiments using the Efficient Global Optimization framework. Results of the EGO experiments show that the heuristic can be used in such optimization settings and that the performance in both high and low dimensions, using different statistical models, can even outperform the original EGO concept that uses Kriging. It also shows that different regression models can be used in Efficient Global Optimization using such a heuristic as prediction variance, making EGO more widely applicable. It is shown that the proposed heuristic is robust with respect of its parameter k, the number of neighbors, and a recommendation of $k = 20$ is given.

7 Conclusions and Outlook

Data science, as in the combination of analytics, machine learning, data mining and also optimization, plays an increasingly important role in industrial processes. Lots of sensor data, machine parameters, quality measurements, images and other signals are collected during these, usually complex, processes. Data mining can aid in extracting relevant information from this data to provide insight to domain experts. In addition, machine learning can aid the domain experts by predicting possible defects, quality indicators or machine parameters. Last but not least, optimization algorithms can be exploited to increase the overall productivity of a manufacturing process, increase the product quality or lower the cost.

Each of these data science components strongly interacts with each other, machine learning requires features which can be collected by data mining algorithms or by using the insight that data mining may provide. The optimization algorithms, on the other hand, can exploit the trained machine learning models to optimize the process efficiently, without requiring lots of evaluations on the real process.

In the context of the PROMIMOOC project these components are presented in a framework for controlling and optimizing industrial manufacturing processes. In Section 7.1 conclusions are drawn from the works in this dissertation and the contributions to the components and interlinking of components of this framework are discussed. Section 7.2 discusses what can be done in the (near) future regarding industrial process optimization.

7.1 Conclusions

To answer the research question, *"How can supervised and unsupervised machine learning techniques be utilized in complex industrial processes?"*, a framework for monitoring, controlling and optimizing an industrial manufacturing process is presented consisting of several components (Chapter 2). The aim of such a framework is three fold. The first objective is to provide the domain expert / process controller with relevant information about the current and possible future states of the process as quickly as possible. Secondly, to warn the domain expert in case of anomalies, and last but not least, to provide the domain expert with a near-optimal set of machine parameters to produce the next batch of products.

For such a framework to work efficiently, not only does it require algorithms with sufficient predictive power, it also requires algorithms that can work with high-dimensional data, in limited time and with a limited number of evaluations (in case of optimization algorithms). In addition, the framework needs to cope with data quality issues such as missing values. To solve this issue and as an answer to the research question, *"How can real-world data issues, such as missing data, be treated efficiently?"* , a novel algorithm for repairing missing values *Incremental Attribute Regression Imputation* (IARI) is proposed. In addition, a visualization technique to get more insight into missing value patterns is presented.

Outlier detection, or anomaly detection is an important component of the framework, as it is the first component that delivers crucial information to the domain expert. In most manufacturing processes the source material used for the final products come from different suppliers and vary in material properties. Outlier detection on these material properties is not straight-forward. Leading to the research question *"How can unsupervised learning be used for anomaly detection in high-dimensional industrial applications?"*. *Global Local Outliers in SubSpaces* (GLOSS) is specifically proposed to find outliers in high-dimensional data sets from a mixed distribution. GLOSS is able to find many relevant outliers in these material property data sets that existing state-of-the-art outlier algorithms are not able to find. By combining local and global neighborhood information, GLOSS works even when the dimensionality is huge and the outlier only shows outlying behaviour in a small subset of these dimensions.

To provide the domain expert with possible machine parameters to handle the incoming material, data driven models and optimization algorithms are required. One of the most used global optimization algorithms for expensive (in time or money) function evaluations, such as an expensive industrial process or time consuming simulator, is *Efficient Global Optimization* (EGO). EGO exploits an underlying Kriging model to find the next candidate solution, effectively replacing the expensive objective function with a much cheaper (but less accurate) surrogate model. The candidate solution is then evaluated on the expensive function and the result is used to update the underlying model. EGO uses the Kriging model because it does not only provide a predicted mean but also a prediction variance, the so-called Kriging variance. Unfortunately, Kriging has a major disadvantage, the training time complexity of the model is $O(n^3)$ and the memory complexity $O(n^2)$, which makes the model quickly infeasible as the data size increases. Leading to the two research questions *"How can predictive models such as Kriging be efficiently applied to a large amount of data?"* and the larger question; *"How can global optimization be applied in an industrial context, even in real-time?"*.

An answer and solution to these research questions, is the proposed *Cluster Kriging*. Cluster Kriging is a Kriging approximation algorithm where the data is being split into clusters or partitions. On each of these clusters a normal Kriging model is being trained. Using either an optimal weight scheme or membership probabilities, the different Kriging models' predictions and variances are combined into one final prediction and prediction variance. Using Cluster Kriging the time complexity of training the model goes down by a factor of k^2, where k is the number of clusters. When k is set proportional to the number of data points and k CPUs are provided, the training of the model even becomes linear. It is shown that the accuracy of Cluster Kriging is consistent and sometimes even outperforming the original Kriging algorithm on a large set of functions.

Another solution to the problem of the EGO with Kriging complexity is to not use Kriging as the underlying surrogate model. Unfortunately, many machine learning techniques do not have a prediction variance or prediction uncertainty available out of the box. In this dissertation an uncertainty measure is proposed, the k-NN uncertainty. This heuristic uncertainty measure is model independent and calculates an approximation of the prediction error using the error information of the

k-nearest neighbors. It is shown that EGO in combination with different models, such as artificial neural networks, support vector machines and random forests, using the k-NN uncertainty, performs equally well or better than using the original EGO with Kriging. The performance of the EGO algorithm is very dependent on the fit of the underlying model, being able to use different predictive models allows for choosing a better fitting model and therefor increasing the efficiency of the EGO algorithm.

7.2 Future work

This dissertation contributes by providing a set of specific solutions in the fields of missing value imputation, anomaly detection, Kriging and model-based optimization. In addition, an overview of a framework for monitoring and optimizing industrial manufacturing processes by using data mining, machine learning and optimization techniques is provided. Different implementations of the framework components are suggested and several issues in connecting these components are challenged. A lot of challenges are of course still present and the design choices made during this project will not always be optimal for each application.

Some of the open questions related to the research in this dissertation are:

Proper benchmarking of outlier detection algorithms: One of the most challenging topics in outlier detection is that the result of an outlier detection algorithm, most of the time, cannot be evaluated using a simple metric. There is no one definition of what an outlier or anomaly is. How a domain expert defines an anomaly depends highly on the domain, application and on the expert. To compare outlier detection algorithms, well defined benchmark data sets are essential. In current state-of-the-art research, classification data sets are misused to validate the performance of anomaly detection algorithms by detecting the least present class in the data set. This results in a biased comparison which favors clustering based algorithms that use the residual class as outlier identification. The development of a proper benchmark for these algorithms, including a wide variety of anomaly and data types, would aid the research field enormously.

Automatically configuring predictive models: The predictive models such as Cluster Kriging, Deep Neural Networks and Random Forests all have a few to many hyper-parameters. In the development of the framework for Tata Steel and BMW, most of these models' parameters where optimized by hand, grid search or simply by taking the most commonly used values from literature. Algorithm configuration is an upcoming research field that could provide very valuable insight and improvements in selecting and configuring these models for optimal performance.

Automatically optimizing artificial neural network architectures: Next to the hyper-parameters of the more classical predictive models, deep neural networks consist of many design choices. From the number of layers to the hyper-parameters of each layer, the preprocessing step of the input data and the activation functions. All of these design choices could possibly be automated by using efficient optimization techniques. Reducing the effort of finding a good predictive model and possibly increasing the final accuracy of these predictive models.

A

IARI results

In this appendix the complete results of the IARI experiments are provided, each section presents the results of a particular dataset from the UCI repository. The third colomn in each table presents the reference value (R^2 score on the dataset without missing values), the highest R^2 score in each row is printed bold.

Cover Type Dataset Results

In Tables A.1 to A.3 and A.4 the accuracy of the models (Accuracy Score) and the quality of imputation ($RMSE$) are shown for the imputation algorithms on 40.000 instances of the Cover Type dataset.

Table A.1: Model Accuracy Score on the Cover Type Dataset with 40.000 instances using Random Forests

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
4	MNAR	0.911	0.887	0.879	0.876	0.886	0.886	**0.893**
6	MNAR	0.911	0.871	0.864	0.860	0.868	0.868	**0.881**
8	MNAR	0.911	0.850	0.841	0.835	0.844	0.845	**0.864**
10	MNAR	0.911	0.815	0.809	0.803	0.806	0.805	**0.839**
12	MNAR	0.911	0.670	0.678	0.656	0.657	0.663	**0.693**
10	MAR	0.911	0.893	0.899	0.900	0.900	0.896	**0.907**
20	MAR	0.911	0.874	0.887	0.886	0.883	0.880	**0.899**
30	MAR	0.911	0.857	0.874	0.874	0.866	0.863	**0.889**
40	MAR	0.911	0.834	0.859	0.858	0.845	0.845	**0.878**
50	MAR	0.911	0.808	0.841	0.838	0.819	0.822	**0.864**
60	MAR	0.911	0.776	0.824	0.822	0.787	0.799	**0.847**

Table A.2: Model Accuracy Score on the Cover Type Dataset with 40.000 instances using Support Vector Machines

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
4	MNAR	0.819	0.746	0.741	0.724	0.746	0.747	0.748
6	MNAR	0.819	0.723	0.717	0.696	0.723	0.727	**0.736**
8	MNAR	0.819	0.704	0.689	0.675	0.704	0.708	**0.735**
10	MNAR	0.819	0.672	0.666	0.662	0.671	0.678	**0.731**
12	MNAR	0.819	0.613	**0.621**	0.593	0.612	0.612	0.602
10	MAR	0.819	0.813	0.813	0.810	0.815	0.815	**0.819**
20	MAR	0.819	0.804	0.806	0.799	0.807	0.810	**0.819**
30	MAR	0.819	0.795	0.797	0.785	0.797	0.803	**0.817**
40	MAR	0.819	0.787	0.788	0.769	0.788	0.794	**0.814**
50	MAR	0.819	0.778	0.778	0.755	0.779	0.784	**0.809**
60	MAR	0.819	0.765	0.764	0.747	0.767	0.776	**0.802**

Table A.3: Model Accuracy Score on the Cover Type Dataset with 40.000 instances using Gradient Boosting Trees

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
4	MNAR	0.787	0.778	0.776	0.761	0.778	0.777	0.779
6	MNAR	0.787	0.774	0.769	0.747	0.772	0.771	**0.778**
8	MNAR	0.787	0.767	0.758	0.745	0.767	0.765	**0.773**
10	MNAR	0.787	0.753	0.748	0.743	0.751	0.750	**0.764**
12	MNAR	0.787	0.695	0.694	**0.717**	0.687	0.690	0.691
10	MAR	0.787	0.785	0.785	0.773	0.785	0.784	0.785
20	MAR	0.787	0.782	0.783	0.771	0.783	0.778	**0.784**
30	MAR	0.787	0.778	0.778	0.765	0.779	0.777	**0.783**
40	MAR	0.787	0.774	0.774	0.763	0.775	0.774	**0.780**
50	MAR	0.787	0.770	0.770	0.760	0.770	0.766	**0.778**
60	MAR	0.787	0.766	0.767	0.755	0.767	0.758	**0.774**

Table A.4: Imputation Quality (RMSE) of each Imputation Algorithm on the Cover Type dataset with 40.000 instances

Miss.%	Type	Mean	Median	Freq.	PVI NN	RI	IARI
4	MNAR	0.755	0.762	0.774	0.755	0.745	**0.721**
6	MNAR	0.786	0.795	0.813	0.786	0.776	**0.760**
8	MNAR	0.814	0.823	0.842	0.814	0.802	**0.771**
10	MNAR	0.848	0.852	0.867	0.847	0.838	**0.791**
12	MNAR	0.894	0.884	0.889	0.894	0.894	**0.877**
10	MAR	0.271	0.277	0.294	0.261	0.225	**0.176**
20	MAR	0.380	0.389	0.414	0.370	0.330	**0.266**
30	MAR	0.468	0.479	0.509	0.460	0.420	**0.347**
40	MAR	0.540	0.552	0.588	0.533	0.496	**0.422**
50	MAR	0.602	0.615	0.654	0.597	0.564	**0.493**
60	MAR	0.661	0.676	0.718	0.658	0.630	**0.564**

Table A.5: Execution time of Imputation Algorithms on the Cover Type Dataset with values 50% MAR in seconds.

Mean	Median	Freq.	PVI NN	RI	IARI
0.03	0.11	0.48	61.47	381.75	119.12

Digits Dataset Results

In Tables A.6 to A.8 the accuracy scores are shown and in Table A.9 the $RMSE$ results are shown for the imputation algorithms on the Page Blocks dataset.

Table A.6: Model Accuracy Score on the Digits Dataset using Random Forests

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
8	MNAR	0.972	0.971	**0.972**	0.966	0.972	0.970	0.972
16	MNAR	0.972	**0.969**	0.967	0.953	0.967	0.967	0.968
23	MNAR	0.972	**0.966**	0.964	0.937	0.962	0.963	0.962
25	MNAR	0.972	**0.967**	0.951	0.904	0.954	0.961	0.947
27	MNAR	0.972	**0.950**	0.923	0.815	0.934	0.932	0.944
10	MAR	0.972	0.969	**0.971**	0.967	0.970	0.971	0.971
20	MAR	0.972	0.964	0.963	0.961	0.967	0.968	**0.970**
30	MAR	0.972	0.962	0.961	0.956	0.963	0.963	**0.967**
40	MAR	0.972	0.954	0.957	0.952	0.959	0.960	**0.962**
50	MAR	0.972	0.950	0.952	0.945	0.953	0.957	**0.957**
60	MAR	0.972	0.944	0.943	0.934	0.944	0.948	**0.953**

Table A.7: Model Accuracy Score on the Digits Dataset using Support Vector Machines

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
8	MNAR	0.980	0.966	0.966	0.964	0.967	0.967	**0.969**
16	MNAR	0.980	0.955	0.952	0.946	0.958	0.958	**0.963**
23	MNAR	0.980	0.943	0.934	0.916	0.943	0.945	**0.954**
25	MNAR	0.980	0.914	0.897	0.841	0.917	0.920	**0.933**
27	MNAR	0.980	0.816	0.754	0.553	0.815	0.823	**0.891**
10	MAR	0.980	0.977	0.978	0.977	0.979	0.978	**0.980**
20	MAR	0.980	0.974	0.973	0.969	0.977	0.976	**0.979**
30	MAR	0.980	0.967	0.966	0.962	0.974	0.971	**0.979**
40	MAR	0.980	0.962	0.960	0.954	0.966	0.965	**0.976**
50	MAR	0.980	0.953	0.950	0.946	0.959	0.958	**0.974**
60	MAR	0.980	0.941	0.938	0.929	0.947	0.947	**0.967**

Table A.8: Model Accuracy Score on the Digits Dataset using Gradient Boosting Trees

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
8	MNAR	0.960	0.960	**0.961**	0.959	0.958	0.958	0.957
16	MNAR	0.960	**0.959**	0.954	0.945	0.956	0.954	0.956
23	MNAR	0.960	**0.956**	0.949	0.934	0.952	0.953	0.943
25	MNAR	0.960	0.949	0.936	0.892	0.940	**0.950**	0.918
27	MNAR	0.960	0.915	0.858	0.746	0.882	0.901	**0.924**
10	MAR	0.960	0.962	**0.962**	0.960	0.958	0.960	0.959
20	MAR	0.960	0.957	0.958	0.956	0.955	**0.960**	0.956
30	MAR	0.960	0.954	0.952	0.953	**0.958**	0.953	0.953
40	MAR	0.960	0.950	0.949	0.944	0.949	**0.952**	0.947
50	MAR	0.960	**0.944**	0.944	0.941	0.944	0.944	0.944
60	MAR	0.960	0.937	0.938	0.928	0.937	0.934	0.930

Table A.9: Imputation Quality (RMSE) of each Imputation Algorithm on the Digits Dataset

Miss.%	Type	Mean	Median	Freq.	PVI NN	RI	IARI
8	MNAR	0.499	0.524	0.577	0.464	0.466	**0.419**
16	MNAR	0.608	0.649	0.752	0.566	0.565	**0.479**
23	MNAR	0.723	0.775	0.919	0.691	0.680	**0.534**
25	MNAR	0.858	0.903	1.037	0.841	0.829	**0.646**
27	MNAR	0.974	0.994	1.103	0.960	0.963	**0.850**
10	MAR	0.265	0.279	0.358	0.193	0.211	**0.154**
20	MAR	0.380	0.399	0.511	0.299	0.316	**0.231**
30	MAR	0.462	0.486	0.623	0.384	0.395	**0.289**
40	MAR	0.537	0.564	0.721	0.470	0.472	**0.345**
50	MAR	0.602	0.632	0.807	0.548	0.543	**0.400**
60	MAR	0.658	0.692	0.883	0.619	0.608	**0.451**

Table A.10: Execution time of Imputation Algorithms on the Digits Dataset with values 50% MAR in seconds.

Mean	Median	Freq.	PVI NN	RI	IARI
0.00	0.01	0.02	9.21	30.39	14.25

Houses 16H Dataset Results

In Tables A.11 to A.13 the R^2 scores are shown and in Table A.15 the $RMSE$ results are shown for the imputation algorithms on the Houses 16H dataset.

Table A.11: Model Accuracy Score (R^2) on the Houses Dataset using Random Forests

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
10	MNAR	0.636	0.619	0.622	0.606	0.623	0.621	**0.630**
20	MNAR	0.636	0.604	0.598	0.580	0.606	0.603	**0.617**
30	MNAR	0.636	0.582	0.561	0.545	0.575	0.574	0.584
40	MNAR	0.636	0.534	0.491	0.485	0.511	0.520	0.531
49	MNAR	0.636	−0.277	−0.287	−0.545	−0.405	**−0.171**	−0.450
10	MAR	0.636	0.624	0.620	0.610	0.624	0.621	**0.627**
20	MAR	0.636	0.604	0.599	0.586	0.610	0.598	**0.620**
30	MAR	0.636	0.577	0.571	0.555	0.586	0.565	**0.608**
40	MAR	0.636	0.544	0.533	0.511	0.552	0.521	**0.590**
50	MAR	0.636	0.499	0.483	0.450	0.519	0.485	**0.567**
60	MAR	0.636	0.423	0.402	0.375	0.458	0.414	**0.536**

Table A.12: Model Accuracy Score (R^2) on the Houses Dataset using Support Vector Machines

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
10	MNAR	0.531	0.502	0.506	0.495	0.508	0.511	**0.522**
20	MNAR	0.531	0.472	0.477	0.469	0.478	0.485	**0.510**
30	MNAR	0.531	0.443	0.446	0.441	0.445	0.455	**0.493**
40	MNAR	0.531	0.403	0.404	0.403	0.402	0.414	**0.469**
49	MNAR	0.531	−0.342	−0.265	**−0.296**	−0.359	−0.341	−0.714
10	MAR	0.531	0.505	0.511	0.505	0.512	0.514	**0.526**
20	MAR	0.531	0.482	0.496	0.492	0.493	0.499	**0.522**
30	MAR	0.531	0.459	0.477	0.476	0.468	0.478	**0.517**
40	MAR	0.531	0.436	0.460	0.459	0.444	0.460	**0.511**
50	MAR	0.531	0.409	0.439	0.438	0.417	0.436	**0.506**
60	MAR	0.531	0.384	0.419	0.417	0.391	0.413	**0.495**

Table A.13: Model Accuracy Score (R^2) on the Houses Dataset using Gradient Boosting Trees

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
10	MNAR	0.582	0.574	0.577	0.563	0.575	0.574	0.577
20	MNAR	0.582	0.566	0.567	0.553	0.563	0.563	0.569
30	MNAR	0.582	0.553	0.553	0.541	0.547	0.550	0.551
40	MNAR	0.582	0.537	0.534	0.517	0.525	0.527	0.522
49	MNAR	0.582	−1.576	−1.161	−2.009	−1.060	−1.182	**−0.516**
10	MAR	0.582	0.568	0.574	0.565	0.574	0.572	**0.577**
20	MAR	0.582	0.559	0.562	0.551	0.561	0.559	**0.575**
30	MAR	0.582	0.547	0.548	0.536	0.550	0.544	**0.568**
40	MAR	0.582	0.530	0.531	0.515	0.532	0.525	**0.561**
50	MAR	0.582	0.510	0.507	0.490	0.512	0.500	**0.547**
60	MAR	0.582	0.486	0.483	0.464	0.490	0.471	**0.527**

Table A.14: Execution time of Imputation Algorithms on the Houses Dataset with values 50% MAR in seconds.

Mean	Median	Freq.	PVI NN	RI	IARI
0.01	0.03	3.28	16.69	96.56	48.62

Table A.15: Imputation Quality (RMSE) of each Imputation Algorithm on the Houses Dataset

Miss.%	Type	Mean	Median	Freq.	PVI NN	RI	IARI
10	MNAR	0.329	0.352	0.498	0.324	0.277	**0.228**
20	MNAR	0.486	0.517	0.709	0.485	0.428	**0.342**
30	MNAR	0.630	0.662	0.869	0.632	0.580	**0.452**
40	MNAR	0.785	0.801	1.007	0.787	0.753	**0.587**
49	MNAR	0.956	0.925	1.134	0.955	0.954	**0.927**
10	MAR	0.270	0.276	0.382	0.253	0.222	**0.190**
20	MAR	0.386	0.395	0.542	0.370	0.328	**0.280**
30	MAR	0.475	0.486	0.664	0.462	0.415	**0.352**
40	MAR	0.545	0.558	0.764	0.535	0.487	**0.412**
50	MAR	0.607	0.622	0.850	0.600	0.555	**0.466**
60	MAR	0.669	0.685	0.925	0.665	0.625	**0.531**

Page Blocks Dataset

In Tables A.16 to A.18 the accuracy scores are shown and in Table A.20 the $RMSE$ results are shown for the imputation algorithms on the Page Blocks dataset.

Table A.16: Model Accuracy Score on the Page Dataset using Random Forests

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
10	MNAR	0.973	0.973	0.973	0.973	**0.974**	0.974	0.973
20	MNAR	0.973	0.972	0.973	0.972	**0.973**	0.972	0.972
30	MNAR	0.973	0.971	0.972	0.971	0.972	**0.974**	0.971
40	MNAR	0.973	0.971	0.970	0.970	0.971	**0.972**	0.969
49	MNAR	0.973	0.960	0.959	**0.960**	0.957	0.949	0.945
10	MAR	0.973	0.974	0.973	**0.974**	0.973	0.974	0.973
20	MAR	0.973	0.974	0.972	**0.974**	0.974	0.974	0.974
30	MAR	0.973	0.972	0.972	0.973	**0.973**	0.972	0.973
40	MAR	0.973	0.971	0.972	0.971	0.973	0.971	**0.973**
50	MAR	0.973	0.972	0.971	0.969	0.970	0.968	**0.972**
60	MAR	0.973	0.969	0.968	0.968	0.969	0.966	**0.970**

Table A.17: Model Accuracy Score on the Page Dataset using Support Vector Machines

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
10	MNAR	0.960	0.959	0.958	0.953	0.959	0.959	**0.960**
20	MNAR	0.960	0.958	0.957	0.952	0.958	0.959	**0.960**
30	MNAR	0.960	0.956	0.952	0.951	0.955	0.957	**0.959**
40	MNAR	0.960	0.945	0.943	0.944	0.945	0.949	**0.955**
49	MNAR	0.960	0.888	0.889	**0.895**	0.888	0.888	0.892
10	MAR	0.960	0.959	0.959	0.954	0.959	0.960	**0.960**
20	MAR	0.960	0.957	0.958	0.952	0.959	0.960	**0.960**
30	MAR	0.960	0.955	0.956	0.950	0.956	0.960	**0.960**
40	MAR	0.960	0.952	0.954	0.946	0.953	0.958	**0.961**
50	MAR	0.960	0.950	0.951	0.942	0.951	0.957	**0.959**
60	MAR	0.960	0.946	0.945	0.937	0.946	0.954	**0.960**

Table A.18: Model Accuracy Score on the Page Dataset using Gradient Boosting Trees

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
10	MNAR	0.971	0.971	0.972	0.970	0.971	**0.972**	0.971
20	MNAR	0.971	0.970	0.970	0.971	0.972	0.971	**0.972**
30	MNAR	0.971	0.970	0.969	0.969	0.971	**0.971**	0.970
40	MNAR	0.971	0.967	0.966	0.965	0.966	**0.970**	0.969
49	MNAR	0.971	0.943	**0.948**	0.942	0.939	0.943	0.930
10	MAR	0.971	**0.973**	0.972	0.971	0.972	0.972	0.971
20	MAR	0.971	0.972	**0.973**	0.970	0.971	0.972	0.971
30	MAR	0.971	0.971	0.971	0.970	**0.972**	0.970	0.971
40	MAR	0.971	0.968	0.968	0.967	0.969	0.968	**0.972**
50	MAR	0.971	0.966	0.964	0.964	0.967	0.967	**0.969**
60	MAR	0.971	0.961	0.959	0.962	0.962	0.965	**0.967**

Table A.19: Execution time of Imputation Algorithms on the Page Dataset with values 50% MAR in seconds.

Mean	Median	Freq.	PVI NN	RI	IARI
0.00	0.00	0.10	1.97	35.32	19.89

Table A.20: Imputation Quality (RMSE) of each Imputation Algorithm on the Page Dataset

Miss.%	Type	Mean	Median	Freq.	PVI NN	RI	IARI
10	MNAR	0.384	0.407	0.473	0.350	0.307	**0.220**
20	MNAR	0.543	0.574	0.649	0.517	0.455	**0.282**
30	MNAR	0.682	0.714	0.710	0.668	0.605	**0.389**
40	MNAR	0.816	0.838	0.819	0.813	0.771	**0.496**
49	MNAR	0.978	0.965	**0.939**	0.977	0.976	0.946
10	MAR	0.276	0.284	0.429	0.240	0.207	**0.126**
20	MAR	0.403	0.413	0.610	0.367	0.323	**0.230**
30	MAR	0.482	0.495	0.739	0.452	0.399	**0.287**
40	MAR	0.563	0.578	0.860	0.537	0.481	**0.317**
50	MAR	0.634	0.651	0.964	0.616	0.555	**0.367**
60	MAR	0.688	0.707	1.051	0.673	0.617	**0.417**

Concrete Dataset

In Tables A.21 to A.23 the R^2 scores are shown and in Table A.25 the $RMSE$ results are shown for the imputation algorithms on the Concrete dataset.

Table A.21: Model Accuracy (R^2) Score on the Concrete Dataset using Random Forests

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
10	MNAR	0.906	**0.897**	0.886	0.880	0.892	0.893	0.891
20	MNAR	0.906	**0.883**	0.867	0.860	0.873	0.875	0.877
29	MNAR	0.906	**0.841**	0.839	0.839	0.824	0.823	0.825
39	MNAR	0.906	**0.821**	0.809	0.816	0.801	0.796	0.762
47	MNAR	0.906	0.727	0.720	0.726	0.712	0.701	0.642
10	MAR	0.906	0.884	0.879	0.877	0.885	0.880	**0.892**
20	MAR	0.906	0.866	0.859	0.852	0.866	0.855	**0.877**
30	MAR	0.906	0.849	0.842	0.833	0.849	0.828	**0.862**
40	MAR	0.906	0.824	0.821	0.812	0.830	0.795	**0.844**
50	MAR	0.906	0.791	0.787	0.775	0.799	0.747	**0.819**
60	MAR	0.906	0.759	0.760	0.752	0.762	0.694	**0.792**

Table A.22: Model Accuracy (R^2) Score on the Concrete Dataset using Support Vector Machines

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
10	MNAR	0.637	0.610	0.597	0.596	0.608	0.622	**0.630**
20	MNAR	0.637	0.564	0.549	0.542	0.556	0.583	**0.609**
29	MNAR	0.637	0.482	0.478	0.472	0.478	0.498	**0.541**
39	MNAR	0.637	0.409	0.398	0.391	0.406	0.433	**0.529**
48	MNAR	0.637	0.240	0.257	0.269	0.237	0.244	**0.318**
10	MAR	0.637	0.619	0.618	0.606	0.622	0.628	**0.634**
20	MAR	0.637	0.597	0.596	0.574	0.603	0.615	**0.631**
30	MAR	0.637	0.574	0.572	0.535	0.578	0.598	**0.628**
40	MAR	0.637	0.552	0.549	0.502	0.555	0.584	**0.625**
50	MAR	0.637	0.526	0.519	0.453	0.528	0.563	**0.620**
60	MAR	0.637	0.490	0.484	0.412	0.493	0.535	**0.610**

Table A.23: Model Accuracy (R^2) Score on the Concrete Dataset using Gradient Boosting Trees

Miss.%	Type	Ref.	Mean	Median	Freq.	PVI NN	RI	IARI
10	MNAR	0.899	0.895	0.889	0.889	0.891	0.892	0.895
20	MNAR	0.899	**0.890**	0.872	0.869	0.881	0.880	0.883
29	MNAR	0.899	0.829	0.842	**0.844**	0.822	0.816	0.833
39	MNAR	0.899	0.820	0.824	**0.827**	0.810	0.803	0.794
47	MNAR	0.899	0.701	0.719	**0.727**	0.694	0.689	0.637
10	MAR	0.899	0.889	0.889	0.881	0.887	0.883	**0.893**
20	MAR	0.899	0.873	0.871	0.864	0.870	0.865	**0.882**
30	MAR	0.899	0.851	0.856	0.844	0.854	0.845	**0.870**
40	MAR	0.899	0.834	0.835	0.827	0.832	0.822	**0.853**
50	MAR	0.899	0.802	0.807	0.791	0.808	0.793	**0.831**
60	MAR	0.899	0.765	0.767	0.759	0.767	0.755	**0.808**

Table A.24: Execution time of Imputation Algorithms on the Concrete Dataset with values 50% MAR in seconds.

Mean	Median	Freq.	PVI NN	RI	IARI
0.00	0.00	0.01	0.15	2.79	1.99

Table A.25: Imputation Quality (RMSE) of each Imputation Algorithm on the Concrete Dataset

Miss.%	Type	Mean	Median	Freq.	PVI NN	RI	IARI
10	MNAR	0.343	0.400	0.469	0.331	0.271	**0.178**
20	MNAR	0.516	0.591	0.667	0.514	0.440	**0.292**
29	MNAR	0.689	0.775	0.831	0.691	0.628	**0.447**
39	MNAR	0.860	0.905	0.963	0.862	0.818	**0.589**
47	MNAR	1.058	1.036	1.092	1.058	1.055	**1.015**
10	MAR	0.277	0.295	0.378	0.240	0.209	**0.140**
20	MAR	0.389	0.415	0.527	0.360	0.311	**0.213**
30	MAR	0.477	0.509	0.653	0.456	0.399	**0.280**
40	MAR	0.549	0.585	0.746	0.534	0.476	**0.342**
50	MAR	0.613	0.653	0.843	0.603	0.547	**0.410**
60	MAR	0.674	0.717	0.909	0.668	0.619	**0.476**

Symbols and Abbreviations

Symbols

Notation. Throughout this Thesis, we used the following common symbols to denote the respective variables in the table below unless otherwise noted.

Symbol	Description
n	Number of data points
k	Number of clusters or neighborhood size
d	Dimensionality of the input space
D	Data set
w	Weight
x	Data point
X	Set of data points
y	Observation
Y	Set of observations
R^2	Coefficient of Determination
$\mathcal{F}$	Subspaces
$NN_k(\cdot)$	k-Nearest Neighbors
$f(\cdot)$	Function
$\hat{f}(\cdot)$	Predicted function (Predictor)
$k(\cdot, \cdot)$	Kernel function
$d(\cdot, \cdot)$	Distance metric
$Pr(\cdot\|\cdot)$	Probability distribution
σ	Standard deviation
σ^2	Variance
μ	Mean values
$f : \mathbb{R}^d \to \mathbb{R}$	Regression function

Abbreviations

Abb.	Full name
AUC	Area Under the Curve
BCM	Bayesian Committee Machines
BFGS	Broyden Fletcher Goldfarb Shanno
BLUP	Best Linear Unbiased Predictor
CCA	Complete Case Analysis
CCPP	Combined Cycle Power Plant
CK	Cluster Kriging
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CWI	Centrum Wiskunde & Informatica
DACE	Design and Analysis of Computer Experiments
DOE	Design of Experiments
ETL	Extraction, transferring and loading
EGO	Efficient Global Optimization
EM	Expectation Maximization
EI	Expected Improvement
FCM	Fuzzy C-Means
FITC	Fully Independent Training Conditional
FM	Finishing Mill
GLOSS	Global Local Outliers in Subspaces
GMM	Gaussian Mixture Models
GMMCK	Gaussian Mixture Model Cluster Kriging
GPR	Gaussian Process Regression
HiCS	High Contrast Subspaces
HSM2	Hot Strip Mill 2
IARI	Incremental Attribute Regression Imputation
IMPOC	Impulse Magnetic Process On-line Controller
k-NN	k Neirest Neighbors
LB	Lower Bound
LOCI	Local Correlation Integral
LOF	Local Outlier Factor
LoOP	Local Outlier Probabilities
MAR	Missing At Random
MCAR	Missing Completely At Random
MI	Multiple Imputation
MIV	Missing Indicator variables
MLE	Maximum Likelihood Estimation

Abb.	Full name
MMP	Monotone Mixture Pattern
MNAR	Missing Not At Random
MSLL	Mean Standardized Log Loss
MTCK	Model Tree Cluster Kriging
OGP	Sparse On-Line Gaussian Processes
OK	Ordinary Kriging
OutRank	Outlier Ranking
OWCK	Optimally Weighted Cluster Kriging
OWFCK	Optimally Weighted Fuzzy Cluster Kriging
PI	Probability of Improvement
PLOF	Probabilistic Local Outlier Factor
PROMIMOOC	Process Mining for Multi- Objective Online Control
PVI	Predictive Value Imputation
RI	Regression Imputation
RM	Rougher Mill
RMSE	Root Mean Squared Error
ROC	Receiver Operating Characteristic
RTO	Real Time Optimization
SMSE	Standardized Mean Squared Error
SOD	Subspace Outlier Detection
SoD	Subset of Data
SoR	Subset of Regressors
UK	Universal Kriging

Bibliography

[1] Dimitri P Bertsekas. *Dynamic programming and optimal control*. Vol. 1. 2. Athena Scientific Belmont, MA, 1995.

[2] Ernest Bruce Lee and Lawrence Markus. *Foundations of optimal control theory*. Tech. rep. DTIC Document, 1967.

[3] Kemin Zhou, John Comstock Doyle, Keith Glover, et al. *Robust and optimal control*. Vol. 40. Prentice hall New Jersey, 1996.

[4] Tianyou Chai. "Optimal operational control for complex industrial processes". In: *IFAC Proceedings Volumes* 45.15 (2012), pp. 722–731. DOI: 10.1016/j.arcontrol.2014.03.005.

[5] Kartik B Ariyur and Miroslav Krstic. *Real-time optimization by extremum-seeking control*. John Wiley & Sons, 2003.

[6] Sebastian Engell. "Feedback control for optimal process operation". In: *Journal of process control* 17.3 (2007), pp. 203–219. DOI: 10.1016/j.jprocont.2006.10.011.

[7] Thidarat Tosukhowong et al. "An introduction to a dynamic plant-wide optimization strategy for an integrated plant". In: *Computers & chemical engineering* 29.1 (2004), pp. 199–208.

[8] Stephan Purr et al. "Stamping Plant 4.0–Basics for the Application of Data Mining Methods in Manufacturing Car Body Parts". In: *Key Engineering Materials*. Vol. 639. Trans Tech Publ. 2015, pp. 21–30. DOI: 10.4028/www.scientific.net/kem.639.21.

[9] MS Benmoussat et al. "Automatic metal parts inspection: Use of thermographic images and anomaly detection algorithms". In: *Infrared Physics & Technology* 61 (2013), pp. 68–80. DOI: 10.1016/j.infrared.2013.07.007.

[10] Maarten Vermeij et al. "Monetdb, a novel spatial columnstore dbms". In: *Academic Proceedings of the 2008 Free and Open Source for Geospatial (FOSS4G) Conference, OSGeo*. 2008, pp. 193–199.

[11] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey". In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–15.

[12] Klaus Herrmann and Matthias Irle. "IMPOC©: An Online Material". In: *Flat-Rolled Steel Processes: Advanced Technologies* (2009), pp. 265–271. DOI: 10.1201/9781420072938-c24.

[13] Bas van Stein and Wojtek Kowalczyk. "An Incremental Algorithm for Repairing Training Sets with Missing Values". In: *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*. Springer International Publishing. 2016, pp. 175–186. DOI: 10.1007/978-3-319-40581-0_15.

[14] Bas van Stein, Wojtek Kowalczyk, and Thomas Bäck. "Analysis and Visualization of Missing Value Patterns". In: *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*. Springer International Publishing. 2016, pp. 187–198. DOI: 10.1007/978-3-319-40581-0_16.

[15] D. B. Rubin. "Inference and Missing Data". In: *Biometrika* 63.3 (Dec. 1976), p. 581. ISSN: 00063444. DOI: 10.2307/2335739.

[16] R. J. A. Little and D. B. Rubin. *Statistical analysis with missing data*. 2nd Edition. Hoboken N. J. Wiley, 2002.

[17] Kamakshi Lakshminarayan, Steven a. Harp, and Tariq Samad. "Imputation of missing data in industrial databases". In: *Applied Intelligence* 11 (1999), pp. 259–275. ISSN: 0924669X. DOI: 10.1023/A:1008334909089.

[18] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2009.

[19] D. C. Howell. *The analysis of missing data*. London: Sage, 2007, pp. 208–224.

[20] J. R. Carpenter and M. G. Kenward. *Multiple imputation and its application*. 1st. Wiley, 2013.

[21] S. Greenland and W.D. Finkle. "A critical look at methods for handling missing covariates in epidemiologic regression analyses". In: *American journal of epidemiology* 142.12 (1995). DOI: 10.1093/oxfordjournals.aje.a117592.

[22] Donald B Rubin. *Multiple imputation for nonresponse in surveys*. Vol. 81. John Wiley & Sons, 2004.

[23] Shaun R Seaman and Ian R White. "Review of inverse probability weighting for dealing with missing data." In: *Statistical methods in medical research* 22.3 (June 2013), pp. 278–95. ISSN: 1477-0334. DOI: 10.1177/0962280210395740.

[24] Leo Breiman. "Random Forests". In: *Machine Learning* 36.1/2 (1999), pp. 85–103. DOI: 10.1023/a:1007563306331. URL: https://doi.org/10.1023%2Fa%3A1007563306331.

[25] K. Bache and M. Lichman. *UCI Machine Learning Repository*. http://archive.ics.uci.edu/ml. 2013.

[26] *PASCAL Machine Learning Benchmarks Repository - mldata.org*. http://mldata.org/repository/data.

[27] A. J. Henry et al. "Comparative methods for handling missing data in large databases." In: *Journal of vascular surgery* 58.5 (Nov. 2013), 1353–1359.e6. ISSN: 1097-6809. DOI: 10.1016/j.jvs.2013.05.008.

[28] H. Junninen et al. "Methods for imputation of missing values in air quality data sets". In: *Atmospheric Environment* 38.18 (June 2004), pp. 2895–2907. ISSN: 13522310. DOI: 10.1016/j.atmosenv.2004.02.026.

[29] D. B. Rubin and N. Schenker. "Multiple imputation in healthcare databases: An overview and some applications". In: *Statistics in medicine* 10 (1991), pp. 585–598. DOI: 10.1002/sim.4780100410.

[30] I. Žliobaitė and J. Hollmén. "Optimizing regression models for data streams with missing values". In: *Machine Learning* 99.1 (2015), pp. 47–73. DOI: 10.1007/s10994-014-5450-3.

[31] van Stein, B. *Incremental Attribute Regression Imputation*. https://basvanstein.github.io/IARI/. 2015.

[32] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[33] van Stein, B. *Visualisation tools for data with missing values*. https://github.com/Basvanstein/MissingDataVis. 2015.

[34] Bas van Stein, Matthijs van Leeuwen, and Thomas Bäck. "Local subspace-based outlier detection using global neighbourhoods". In: *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE. 2016, pp. 1136–1142. DOI: 10.1109/bigdata.2016.7840717.

[35] Victoria J Hodge and Jim Austin. "A survey of outlier detection methodologies". In: *Artificial Intelligence Review* 22.2 (2004), pp. 85–126. DOI: 10.1007/s10462-004-4304-y.

[36] V Bamnett and T Lewis. "Outliers in statistical data". In: *Journal of the Royal Statistical Society* 141.4 (1978).

[37] Markus M. Breunig et al. "LOF: Identifying Density-Based Local Outliers". In: *ACM SIGMOD Record* 29.2 (2000), pp. 93–104. ISSN: 01635808. DOI: 10.1145/335191.335388. URL: http://portal.acm.org/citation.cfm?doid=335191.335388.

[38] Spiros Papadimitriou et al. "Loci: Fast outlier detection using the local correlation integral". In: *Data Engineering, 2003. Proceedings. 19th International Conference on* (2003), pp. 315–326. DOI: 10.1109/ICDE.2003.1260802.

[39] Hans-Peter Kriegel et al. "LoOP: local outlier probabilities". In: *Proceedings of the 18th ACM conference on Information and knowledge management* (2009), pp. 1649–1652. DOI: 10.1145/1645953.1646195. URL: http://doi.acm.org/10.1145/1645953.1646195.

[40] Fabian Keller, Emmanuel Muller, and Klemens Bohm. "HiCS: high contrast subspaces for density-based outlier ranking". In: *2012 IEEE 28th International Conference on Data Engineering*. IEEE. 2012, pp. 1037–1048. DOI: 10.1109/icde.2012.88.

[41] Hans-Peter Kriegel et al. "Outlier detection in axis-parallel subspaces of high dimensional data". In: *Advances in Knowledge Discovery and Data Mining*. Springer, 2009, pp. 831–838. DOI: 10.1007/978-3-642-01307-2_86.

[42] H Kriegel et al. "Outlier detection in arbitrarily oriented subspaces". In: *Data Mining (ICDM), 2012 IEEE 12th International Conference on*. IEEE. 2012, pp. 379–388. DOI: 10.1109/icdm.2012.21.

[43] Antonio Loureiro, Luis Torgo, and Carlos Soares. "Outlier detection using clustering methods: a data cleaning application". In: *Proceedings of KD-*

Net Symposium on Knowledge-based Systems for the Public Sector. Bonn, Germany. 2004.

[44] Ville Hautamäki, Ismo Kärkkäinen, and Pasi Fränti. "Outlier Detection Using k-Nearest Neighbour Graph". In: *International Conference on Pattern Recognition (3)*. 2004, pp. 430–433. DOI: 10.1109/icpr.2004.1334558.

[45] Shuchita Upadhyaya and Karanjit Singh. "Classification based outlier detection techniques". In: *International Journal of Computer Trends and Technology* 3.2 (2012), pp. 294–298.

[46] Kokyo Choy. "Outlier detection for stationary time series". In: *Journal of Statistical Planning and Inference* 99.2 (2001), pp. 111–127. DOI: 10.1016/s0378-3758(01)00081-7.

[47] Edwin M Knorr, Raymond T Ng, and Vladimir Tucakov. "Distance-based outliers: algorithms and applications". In: *The VLDB JournalâĂŤThe International Journal on Very Large Data Bases* 8.3-4 (2000), pp. 237–253. DOI: 10.1007/s007780050006.

[48] Feng Chen, Chang-Tien Lu, and Arnold P Boedihardjo. "Gls-sod: a generalized local statistical approach for spatial outlier detection". In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2010, pp. 1069–1078. DOI: 10.1145/1835804.1835939.

[49] Erich Schubert, Arthur Zimek, and Hans-Peter Kriegel. "Local outlier detection reconsidered: a generalized view on locality with applications to spatial, video, and network outlier detection". In: *Data Mining and Knowledge Discovery* 28.1 (2014), pp. 190–237. DOI: 10.1007/s10618-012-0300-z.

[50] Emmanuel Muller et al. "Outlier ranking via subspace analysis in multiple views of the data". In: *Data Mining (ICDM), 2012 IEEE 12th International Conference on*. IEEE. 2012, pp. 529–538. DOI: 10.1109/icdm.2012.112.

[51] Ricardo J. G. B. Campello et al. "Hierarchical Density Estimates for Data Clustering, Visualization, and Outlier Detection". In: *ACM Trans. Knowl. Discov. Data* 10.1 (July 2015), 5:1–5:51. ISSN: 1556-4681. DOI: 10.1145/2733381. URL: http://doi.acm.org/10.1145/2733381.

[52] van Stein, B. *Global Local Outlier Detection in SubSpaces*. https://github.com/Basvanstein/Gloss/. 2015.

[53] Bas van Stein et al. “Cluster-based Kriging Approximation Algorithms for Complexity Reduction”. In: *Data Mining and Knowledge Discovery* (2018), Under review. DOI: 10.1145/3071178.3071321.

[54] Bas van Stein et al. “Fuzzy clustering for optimally weighted cluster kriging”. In: *Fuzzy Systems (FUZZ-IEEE), 2016 IEEE International Conference on*. IEEE. 2016, pp. 939–945. DOI: 10.1109/fuzz-ieee.2016.7737789.

[55] Bas van Stein et al. “Optimally weighted cluster kriging for big data regression”. In: *International Symposium on Intelligent Data Analysis*. Springer, Cham. 2015, pp. 310–321. DOI: 10.1007/978-3-319-24465-5_27.

[56] Hao Wang et al. “Time complexity reduction in efficient global optimization using cluster kriging”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2017, pp. 889–896. DOI: 10.1145/3071178.3071321.

[57] C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*. Adaptive computation and machine learning series. University Press Group Limited, 2006. ISBN: 9780262182539.

[58] Timothy W Simpson et al. “Kriging models for global approximation in simulation-based multidisciplinary design optimization”. In: *AIAA journal* 39.12 (2001), pp. 2233–2241. DOI: 10.2514/3.15017.

[59] Leo Breiman. “Random Forests”. In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32. ISSN: 0885-6125.

[60] A. D’Ambrosio, M. Aria, and R. Siciliano. “Accurate Tree-based Missing Data Imputation and Data Fusion within the Statistical Learning Paradigm”. In: *Journal of classification* 29.2 (2012), pp. 227–258. DOI: 10.1007/s00357-012-9108-1.

[61] Luís Torgo. “Functional models for regression tree leaves”. In: *International Conference on Machine Learning*. Vol. 97. Citeseer. 1997, pp. 385–393.

[62] G Huang, D. H. Wang, and Y. Lan. “Extreme learning machines: a survey”. In: *International Journal of Machine Learning and Cybernetics* 2.2 (May 2011), pp. 107–122. ISSN: 1868-8071. DOI: 10.1007/s13042-011-0019-y.

[63] Vladimir Vapnik. *The nature of statistical learning theory*. Springer Science & Business Media, 2000. DOI: 10.1007/978-1-4757-3264-1.

[64] Martin D Buhmann. "Radial basis functions: theory and implementations". In: *Cambridge monographs on applied and computational mathematics* 12 (2004), pp. 147–165.

[65] V Tresp. "A Bayesian committee machine." In: *Neural computation* 12.11 (2000), pp. 2719–2741. ISSN: 0899-7667. DOI: 10.1162/089976600300014908.

[66] Linda Hartman and Ola Hössjer. "Fast Kriging of large data sets with Gaussian Markov random fields". In: *Computational Statistics & Data Analysis* 52.5 (2008), pp. 2331–2349. ISSN: 01679473. DOI: 10.1016/j.csda.2007.09.018.

[67] Krzysztof Chalupka, Christopher K. I. Williams, and Iain Murray. "A Framework for Evaluating Approximation Methods for Gaussian Process Regression". In: *J. Mach. Learn. Res.* 14.1 (Feb. 2013), pp. 333–350. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2502581.2502592.

[68] Michael L Stein. *Interpolation of spatial data: some theory for Kriging*. Springer Science & Business Media, 1999.

[69] Jack PC Kleijnen. "Kriging metamodeling in simulation: a review". In: *European Journal of Operational Research* 192.3 (2009), pp. 707–716. DOI: 10.2139/ssrn.980063.

[70] Donald R Jones, Matthias Schonlau, and William J Welch. "Efficient global optimization of expensive black-box functions". In: *Journal of Global optimization* 13.4 (1998), pp. 455–492.

[71] Jerome Sacks et al. "Design and analysis of computer experiments". In: *Statistical science* (1989), pp. 409–423.

[72] David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. "Kriging is well-suited to parallelize optimization". In: *Computational Intelligence in Expensive Optimization Problems*. Springer, 2010, pp. 131–162. DOI: 10.1007/978-3-642-10701-6_6.

[73] Neil D Lawrence. "Gaussian process latent variable models for visualisation of high dimensional data". In: *Advances in neural information processing systems* 16.3 (2004), pp. 329–336.

[74] Bernhard W Silverman. "Some aspects of the spline smoothing approach to non-parametric regression curve fitting". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 47.1 (1985), pp. 1–52.

[75] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. "A unifying view of sparse approximate Gaussian process regression". In: *The Journal of Machine Learning Research* 6.1 (2005), pp. 1939–1959.

[76] Lehel Csató and Manfred Opper. "Sparse on-line Gaussian processes". In: *Neural computation* 14.3 (2002), pp. 641–668.

[77] Andrew Naish-Guzman and Sean Holden. "The generalized FITC approximation". In: *Advances in Neural Information Processing Systems*. 2007, pp. 1057–1064.

[78] Edward Snelson and Zoubin Ghahramani. "Sparse Gaussian processes using pseudo-inputs". In: *Advances in neural information processing systems*. 2005, pp. 1257–1264.

[79] Tao Chen and Jianghong Ren. "Bagging for Gaussian Process Regression". In: *Neurocomput.* 72.7-9 (Mar. 2009), pp. 1605–1610. ISSN: 0925-2312.

[80] D. Nguyen-Tuong, M. Seeger, and J. Peters. "Model learning with local Gaussian process regression". In: *Advanced Robotics* 23.15 (2009), pp. 2015–2034. ISSN: 0169-1864. DOI: 10.1163/016918609X12529286896877.

[81] L. Breiman. "Bagging predictors". In: *Machine Learning* 24.2 (1996), pp. 123–140. ISSN: 0885-6125. DOI: 10.1007/BF00058655.

[82] Joseph C Dunn. "A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters". In: *Journal of Cybernetics* 3.3 (1973), pp. 32–57. DOI: 10.1080/01969727308546046.

[83] Douglas Reynolds. "Gaussian mixture models". In: *Encyclopedia of Biometrics*. Springer, 2009, pp. 659–663. DOI: 10.1007/springerreference_70943.

[84] Leo Breiman et al. *Classification and regression trees*. CRC press, 1984. DOI: 10.1201/9781315139470.

[85] Yong Wang and Ian H Witten. *Induction of model trees for predicting continuous classes*. Department of Computer Science, University of Waikato, 1996.

[86] Niels Landwehr, Mark Hall, and Eibe Frank. "Logistic model trees". In: *Machine Learning* 59.1-2 (2005), pp. 161–205. DOI: 10.1007/s10994-005-0466-3.

[87] I-C Yeh. "Modeling of strength of high-performance concrete using artificial neural networks". In: *Cement and Concrete research* 28.12 (1998), pp. 1797–1808.

[88] Heysem Kaya, PÄśnar Tüfekci, and S. Fikret Gürgen. "Local and Global Learning Methods for Predicting Power of a Combined Gas & Steam Turbine". In: *International Conference on Emerging Trends in Computer and Electronics Engineering (ICETCEE 2012)* (2012), pp. 13–18.

[89] Sethu Vijayakumar, Aaron D'souza, and Stefan Schaal. "Incremental online learning in high dimensions". In: *Neural computation* 17.12 (2005), pp. 2602–2634. DOI: 10.1162/089976605774320557.

[90] F. Fortin et al. "DEAP: Evolutionary Algorithms Made Easy". In: *Journal of Machine Learning Research* 13 (July 2012), pp. 2171–2175.

[91] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013. DOI: 10.1002/9781118723203.

[92] Douglas C Montgomery. *Design and analysis of experiments*. John wiley & sons, 2017.

[93] Jonas Mockus. *Bayesian approach to global optimization: theory and applications*. Vol. 37. Springer Science & Business Media, 2012.

[94] J Močkus. "On Bayesian methods for seeking the extremum". In: *Optimization Techniques IFIP Technical Conference*. Springer. 1975, pp. 400–404.

[95] Daniel G. Krige. "A Statistical Approach to Some Basic Mine Valuation Problems on the Witwatersrand". In: *Journal of the Chemical, Metallurgical and Mining Society of South Africa* 52.6 (Dec. 1951), pp. 119–139.

[96] Donald R Jones. "A taxonomy of global optimization methods based on response surfaces". In: *Journal of global optimization* 21.4 (2001), pp. 345–383.

[97] JE Dennis and Virginia Torczon. "Managing approximation models in optimization". In: *Multidisciplinary design optimization: State-of-the-art* (1997), pp. 330–347.

[98] Antanas Žilinskas. "A review of statistical models for global optimization". In: *Journal of Global Optimization* 2.2 (1992), pp. 145–153. DOI: 10.1007/bf00122051.

[99] Jonas Mockus. "Application of Bayesian approach to numerical methods of global and stochastic optimization". In: *Journal of Global Optimization* 4.4 (1994), pp. 347–365. DOI: 10.1007/bf01099263.

[100] Bas van Stein et al. "A Novel Uncertainty Quantification Method for Efficient Global Optimization". In: *Information Processing and Management of Uncertainty in Knowledge-Based Systems. Theory and Foundations*. Springer

International Publishing. 2018, forthcoming. ISBN: 978-3-319-91475-6. DOI: 10.1007/978-3-319-91476-3.

English Summary

Industrial manufacturing processes, such as the production of steel or the stamping of car body parts, are complex semi-batch processes with many process steps, machine parameters and quality indicators. To optimize these complex processes, for example by reducing the number of defects or increasing the throughput, a great number of requirements need to be taken into consideration. In this dissertation a framework for monitoring and optimizing these complex industrial processes is presented. The framework is specifically tailored to the production processes of Tata Steel and BMW Group. Both are industrial partners of the PROMIMOOC project. The framework consists of several components of which; preprocessing, outlier detection, predictive modeling and optimization are the main technical components that are the focus of this work. For each of these components a possible implementation is proposed and the challenges in implementing these components in an industrial manufacturing setting are discussed.

A novel algorithm for repairing missing values, *Incremental Attribute Regression Imputation* (IARI) is proposed and a visualization technique to get more insight into missing value patterns is presented. IARI repairs missing values by training predictive models on (repaired) complete columns (features). The predictive models are then used to repair one column at a time till the dataset is complete. The accuracy of imputation (how well the missing values are repaired) is much better than that of existing imputation techniques.

For the monitoring of the manufacturing process it is vital to detect deviations in source material as quickly as possible, as the quality of the source material can greatly effect the final product quality. An outlier detection algorithm *Global*

Local Outliers in SubSpaces (GLOSS), specifically designed to find outliers in high-dimensional data sets from mixed distributions, is proposed and effectively used on the BMW data set. The proposed algorithm is able to detect "hidden" outliers, outliers that look similar to other data points coming from a different distribution but entirely different from the data points coming from the same distribution. The algorithm combines global distance information, using all dimensions (features), with local subspace distance information (subspace of features) to determine the outlier possibility of each subspace. For the BMW data set, the subspaces are predefined as one percent of the length of a steel coil. In cases where the subspaces are not defined the algorithm employs the interesting subspace search technique used in *High Contrast Subspaces* (HiCS).

To not only monitor but also optimize the production process, predictive models and optimization algorithms are required. One popular algorithm to perform global optimization on expensive black-box functions (such as a complex production process), is the *Efficient Global Optimization* (EGO) algorithm. This algorithm makes use of a surrogate model to minimize the required number of real function evaluations. The surrogate model originally used in EGO is the Kriging, or Gaussian Process regression model. To apply EGO on large high-dimensional data sets, however, is not feasible with the current implementation. Training the underlying Kriging models takes $O(n^3)$ time in the number of data points. To alleviate this issue, a set of Kriging approximation algorithms, *Cluster Kriging* (CK), is proposed. In Cluster Kriging, multiple smaller Kriging models are trained on a subset of the data and then the predictions of these local models are combined using a clever combination technique. Various flavors of Cluster Kriging are implemented and evaluated and it is shown that with Cluster Kriging it becomes not only feasible to model larger data sets, the accuracy of the trained models usually outperform the original Kriging model. This performance gain is likely due to the fact that local information is better preserved in the local Kriging models.

To use the EGO algorithm also with a wide variety of different predictive models, a model-independent uncertainty estimation measure, the k-NN uncertainty, is proposed. Using this heuristic error measure as the prediction variance in EGO, different surrogate models can be used in the EGO framework. It is shown that,

dependent on the data, EGO with a different model such as an artificial neural network, can significantly increase the performance (convergence) of the optimization procedure.

Each algorithm proposed in this thesis is compared against state-of-the-art existing algorithms and used in the PROMIMOOC project for real world applications of BMW and Tata Steel.

Nederlandse Samenvatting

Industriële fabricageprocessen, zoals de productie van staal en het produceren van carrosserie onderdelen (autodeuren, motorkappen etc.), zijn complexe semi-batch processen met veel processtappen, machine instellingen en kwaliteit indicatoren. Om deze complexe processen te optimaliseren, bijvoorbeeld door het aantal defecten te reduceren of door de productie te verhogen, moet aan een flink aantal vereisten voldaan worden. In dit proefschrift wordt een systeem voor het monitoren en optimaliseren van deze complexe industriële processen gepresenteerd. Het systeem is gemaakt voor het productieproces van Tata Steel en BMW, de twee industriële partners van het PROMIMOOC project. Het systeem zal echter ook werken voor een generiek productieproces zolang alle benodigde data aanwezig is. Het systeem bestaat uit een aantal componenten waarvan voorbewerking, "anomaly" detectie, voorspellende modellen en optimalisatie een grote rol spelen en de focus hebben in dit proefschrift. Voor elk van deze componenten wordt er een mogelijke implementatie voorgesteld en worden de uitdagingen op dit gebied in een industriële productiesetting besproken.

Een nieuw algoritme voor het repareren van missende waardes, *Incrementele Attribuut Regressie Imputatie* (IARI) is voorgesteld, als wel een aantal visualisatietechnieken voor het inzichtelijk maken van patronen van missende waardes in een dataset. IARI repareert een dataset door gebruik te maken van voorspellende modellen die gebruik maken van de al bestaande complete en gerepareerde data om zo een voor een kolommen te repareren. Het nieuwe IARI algoritme kan de missende waardes een stuk beter repareren dan andere bestaande state-of-the-art algoritmes.

Voor het monitoren van een productieproces, is het van vitaal belang dat afwijkingen in het te bewerken materiaal snel gedetecteerd worden. Als de kwaliteit van het bronmateriaal niet goed is of grote afwijkingen bevat, kan dit van sterke invloed zijn op het uiteindelijke product. Een anomalie detectie algoritme *Globale Lokale Afwijkingen in Deel Ruimtes* (GLOSS), specifiek ontwikkeld voor het vinden van afwijkingen in datasets met een hoge dimensionaliteit en bestaande uit een mix van verschillende distributies, is voorgesteld en gebruikt op de dataset van BMW. Het voorgestelde algoritme kan zogenaamde "verborgen" afwijkingen vinden. Dit zijn afwijkingen die heel erg lijken op data punten die in de dataset zitten maar van een andere distributie komen en daarentegen zeer afwijken van data punten in de data set die van dezelfde distributie komen. Het algoritme gebruikt afstandinformatie van de volledige hoog-dimensionale ruimte in combinatie met lokale afstandinformatie in deelruimtes, om voor elke deelruimte een afwijk score te berekenen. Voor de BMW dataset zijn de deelruimtes voorgedefinieerd als één procent van de lengte van een rol staal. Als de deelruimtes niet voorgedefinieerd zijn, dan maakt het algoritme gebruik van de interessante deelruimte zoek techniek die ook in het *Hoge Contrast Deel Ruimtes* (HiCS) algoritme wordt gebruikt.

Om niet alleen het proces te monitoren maar ook te optimaliseren, zijn er voorspellende modellen en optimalisatie algoritmes nodig. Een populair algoritme voor globale optimalisatie speciaal voor kostbare (in tijd of geld) functie evaluaties, is het *Efficiënte Globale Optimalisatie* (EGO) algoritme. Dit algoritme maakt gebruik van een surrogaat model om met zo min mogelijk echte evaluaties een zo goed mogelijke oplossing te vinden. Omdat het proberen van instellingen in een productieproces erg duur kan zijn, in zowel tijd als materiaalkosten, is het van belang dat er zo min mogelijk "slechte" instellingen geprobeerd moeten worden voordat een goede oplossing bereikt kan worden. Het surrogaat model dat EGO normaal gesproken gebruikt is Kriging. Kriging is een voorspellend model dat niet alleen de voorspellende waarde geeft, maar ook de waarschijnlijke variantie (mogelijke afwijking) in de voorspelling. Deze variantie wordt ook wel de Kriging variantie genoemd. Deze variantie wordt door het EGO algoritme gebruikt om te balanceren tussen exploitatie en exploratie met een zogenoemd opvul-criterium. Om EGO te gebruiken op een grote dataset met veel attributen, is er echter een groot nadeel, het trainen van een Kriging model kost $O(n^3)$ tijd en $O(n^2)$

geheugen, waarbij n het aantal datapunten is. Om dit probleem op te lossen, stellen we in dit proefschrift een set van Kriging benaderingalgoritmes voor die we *Cluster Kriging* (CK) varianten noemen. In een Cluster Kriging algoritme worden er meerdere kleine Kriging modellen getraind op kleine delen van de dataset. De voorspellingen van deze lokale modellen worden dan op een slimme manier gecombineerd om uiteindelijk tot een voorspelling en variantie te komen. Verschillende varianten van Cluster Kriging worden voorgesteld waarbij elke variant vergeleken wordt in een test opstelling met verschillende test functies. De accuraatheid van de Cluster Kriging varianten is vaak nog beter dan het originele Kriging algoritme. Dit komt waarschijnlijk omdat lokale informatie beter door de lokale modellen opgeslagen kan worden dan in een globaal model.

Een andere oplossing om EGO te gebruiken voor grote datasets is door andere surrogaat modellen te gebruiken. Helaas hebben maar weinig modellen het voordeel dat ze een voorspelling variantie of voorspellingszekerheid geven. In dit werk stellen wij daarom ook een model-onafhankelijke onzekerheid waardering voor, de *k-NN onzekerheid*. Met het gebruik van deze heuristiek kan EGO gebruikt worden met elk mogelijk voorspellend model als surrogaat. We laten zien dat, afhankelijk van de data, EGO met een heel ander model dan Kriging (zoals een neuraal netwerk), significant beter kan werken.

Elk algoritme dat voorgesteld is in dit werk is vergeleken met verschillende state-of-the-art alternatieven. Elk algoritme is ook actief gebruikt gedurende het PROMI-MOOC project binnen in "echte-wereld" applicaties van BMW en Tata Steel.

About the Author

Bas van Stein born 1989 in Sassenheim, The Netherlands, received his VWO degree in 2008 at Rijnlands Lyceum Sassenheim. In 2011, Bas received his Bachelor degree in Informatica (Computer Science) and in 2013 he received his Masters degree in Computer Science, both at the Leiden Institute of Advanced Computer Science (LIACS), Leiden University. During his Bachelor and Master studies, Bas assisted as a student-assistant in a variety of courses and also functioned as a student ambassador, promoting the Informatica Bachelor track by presenting about Computer Science at high schools around the region. Bas started as a Ph.D. student in 2014 at the Leiden University as part of the PROMIMOOC project in the Natural Computing group, under supervision of Thomas Bäck. He specialized in data mining and machine learning and also contributed to the fields of exploratory data mining and Bayesian optimization. During his research period as a Ph.D. student, Bas assisted with the courses: *Concurrency*, *Modeling and Petri Nets*, *Neural Networks*, *Advances in Data Mining* and the *Master Class*, a biweekly meeting of all second year master students in which support is provided for the students to finish their master thesis on time and to prepare them for the job market. Bas gave several practicums for Petri Nets and several lectures for the Neural Networks course.

Next to being a scientist, Bas is also an entrepreneur and has co-founded several companies. In 2012, Bas van Stein started, together with his business partner Tom Groentjes, the company *Van Stein & Groentjes*, a company specialized in custom

tailored software, mobile applications and websites. In 2015 the company transitioned from a V.O.F. legal structure to a B.V. and the first employee was hired. In 2017, Bas participated and co-founded Smartnotation B.V., with the first product, a smart meeting minutes application using artificial intelligence techniques such as voice-to-text. During the same year he co-founded Culture Match B.V., a mobile and web application using recommendation engines to suggest cultural events to users and to suggest companions (friends or strangers) to visit these events with.