

LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT
DEGREE PROGRAM IN COMPUTER SCIENCE

Master's thesis

Modular framework for outlier detection

The topic was accepted in the degree program of Computer Science on 01.04.2014.

Supervisors: PhD Tomi Rätty
 Professor, D.Sc. (Tech) Lasse Lensu

Examiners: Professor, D.Sc. (Tech) Lasse Lensu
 PhD Tomi Rätty

Oulu 2014

Niko Reunanen
niko.reunanen@{vtt,lut}.fi

ABSTRACT

LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT
DEGREE PROGRAM IN COMPUTER SCIENCE

Niko Reunanen

Modular framework for outlier detection

Thesis for the Degree of Master of Science in Technology
2014

104 pages, 25 figures, 20 tables, 4 algorithms and 6 appendices

Examiners: Professor, D.Sc. (Tech) Lasse Lensu
 PhD Tomi Rätty

Keywords: outlier, outlier detection, ensemble learning

Abstract

Outlier detection is an important form of data analysis because outliers in several cases contain the interesting and important pieces of information. In the recent years, many different outlier detection algorithms have been devised for finding different kinds of outliers in varying contexts and environments. Some effort has been put to study how to effectively combine different outlier detection methods. The combination of outlier detection algorithms as an ensemble was studied in this thesis by designing a modular framework for outlier detection, which combines arbitrary outlier detection techniques. This work resulted in an example implementation of the framework. Outlier detection capability of the ensemble method was validated using datasets and methods found in outlier detection research. The framework achieved better results than the individual outlier algorithms. Future research includes how to handle large datasets effectively and the possibilities for real-time outlier monitoring.

TIIVISTELMÄ

LAPPEENRANNAN TEKNILLINEN YLIOPISTO
TUOTANTOTALOUDEN TIEDEKUNTA
TIETOTEKNIKAN KOULUTUSOHJELMA

Niko Reunanen

Modulaarinen menetelmäkehys poikkeamien tunnistamiseen

Diplomityö

2014

104 sivua, 25 kuvaa, 20 taulukkoa, 4 algoritmia ja 6 liitettä

Tarkastajat: Professori, TkT Lasse Lensu
 FT Tomi Rätty

Hakusanat: poikkeama, poikkeaman tunnistaminen, yhdistelmäoppiminen

Tiivistelmä

Poikkeamien tunnistaminen on tärkeä data-analyysin muoto, koska poikkeamat sisältävät useissa tapauksissa mielenkiintoista ja tärkeää informaatiota. Monta erilaista poikkeaman tunnistusalgoritmia on kehitetty viimeisten vuosien aikana löytämään poikkeamia erilaisissa konteksteissa ja ympäristöissä. Poikkeamien tunnistusalgoritmien yhdistämistä on myös hieman tutkittu. Tässä diplomityössä algoritmien yhdistämistä tutkittiin kehittämällä modulaarinen menetelmäkehys poikkeamien tunnistamiseen yhdistämällä valittuja tunnistusalgoritmeja ja toteutettiin esimerkkitoteutus. Menetelmäkehysten suorituskyky validoitiin käyttämällä poikkeamatutkimuksessa esiintyviä menetelmiä ja aineistoja. Toteutuksen suorituskyky oli parempi kuin yksittäisten poikkeamien tunnistusalgoritmien. Jatkotutkimuksen kannalta tutkimusongelmiin sisältyvät esimerkiksi suurten aineistojen tehokas käsittely ja reaaliaikainen poikkeamantunnistaminen.

PREFACE

I want to sincerely thank my supervisors PhD Tomi Rätty and Professor, D.Sc. (Tech) Lasse Lensu for their effort, discussions and patience. Tomi Rätty taught me a lot in the art of scientific writing, reviewed my text extensively and supported me throughout the thesis. Lasse Lensu provided interesting and insightful comments regarding the theoretical content. Thank you both for your valuable time.

This thesis was written at VTT Technical Research Centre of Finland and the thesis is a part of the project "Practical Applications of Model-based technologies to continuous integration and testing methodologies". I want to thank VTT and M.Sc. Mikko Nieminen for hiring me and allowing me to work at VTT.

Oulu, 28.4.2014

Niko Reunanen

Table of contents

1	Introduction	6
1.1	Objectives and scope	7
1.2	Thesis structure	7
2	Related work	9
3	Preliminaries and methods for outlier detection	13
3.1	Outlier types	13
3.1.1	Point anomalies	13
3.1.2	Contextual anomalies	15
3.2	k-Means clustering	17
3.3	Outlier detection algorithms	19
3.3.1	Local Outlier Factor	25
3.3.2	Local Distance-based Outlier Factor	26
3.3.3	Clustering-based Outlier Detection	27
3.4	Principal component analysis	28
3.4.1	Singular value decomposition	30
3.5	Statistical process monitoring	32
3.6	Simulated annealing	34
3.7	Monte Carlo methods	36
4	Modular framework for outlier detection	38
4.1	Framework architecture	42
4.2	Parameter optimization	45
4.3	Data exchange between subprocesses	49
5	MFFOD implementation	52
5.1	Implemented modules	54
6	Characteristics of MFFOD	59
7	Experiments	61
7.1	Wisconsin Breast Diagnostic Cancer	67
7.2	Pen-Based Recognition of Handwritten Digits	71
7.3	Synthetic data	75

7.4 Execution time tests	78
8 Discussion	81
8.1 Future work	83
9 Conclusion	84
REFERENCES	86
APPENDICES	

Abbreviations

BIC	Bayesian Information Criterion
CBOD	Clustering-based Outlier Detection
HeDES	Heterogeneous Detector Ensemble on Random Subspaces
JSON	JavaScript Object Notation
KDE	Kernel Density Estimation
LDOF	Local Density-based Outlier Factor
LOF	Local Outlier Factor
MC	Master Controller
MFFOD	Modular Framework for Outlier Detection
PC	Principal Component
PCA	Principal Component Analysis
SA	Simulated Annealing
SPM	Statistical Process Monitoring
SVD	Singular Value Decomposition

Symbols

i	Matrix cell row index
j	Matrix cell column index
n	Number of data samples, number of rows in matrix \mathbf{X}
d	Number of data dimensions, number of columns in matrix \mathbf{X}
l	Number of principal components
t	Simulated annealing temperature
v	Eigenvector

λ	Eigenvalue
w_k	Ensemble weight coefficient
D	Distance function
Q	Squared prediction error
T^2	Hotelling's T^2
φ	Combined index
δ^2	Squared prediction error control limit
τ^2	Hotelling's T^2 control limit
ς	Combined index control limit
π	Pi
e^x	Exponential function
T	Voting threshold
OF	Outlier Factor
SEN	Sensitivity
SPC	Specificity
TP	Number of true positive detections
TN	Number of true negative detections
FP	Number of false positive detections
FN	Number of false negative detections
\mathbb{R}	Set of all real numbers
χ^2	Chi-square distribution
$\bar{\mathbf{x}}$	Mean value of vector \mathbf{x}
$\sigma(\mathbf{x})$	Standard deviation of vector \mathbf{x}
$ \mathbf{x} $	Number of elements in vector \mathbf{x}

\mathbf{c}_j	Cluster j centroid
\mathbf{x}_i	Data point, the i :th row of matrix \mathbf{X}
\mathbf{E}	Residual matrix
\mathbf{I}	Identity matrix
\mathbf{P}	Loadings matrix
$\tilde{\mathbf{P}}$	Residual loadings matrix
\mathbf{T}	Score matrix
\mathbf{X}	Dataset, matrix
\mathbf{X}^{-1}	Inverse of matrix \mathbf{X}
\mathbf{X}^T	Transpose of \mathbf{X}
$\mathbf{X}\mathbf{X}^T$	Matrix multiplication of \mathbf{X} and \mathbf{X}^T
Σ	Covariance matrix
Σ_{ij}	Covariance between vectors \mathbf{x}_i and \mathbf{x}_j
$\mathcal{N}(\mathbf{x}_i, k)$	Set of k nearest data points of \mathbf{x}_i

1 Introduction

Outlier detection is an important form of data analysis and it has a plethora of uses. A lot of research effort has been put for innovating new outlier detection algorithms and schemes. An outlier is an unexpected data observation that does not match the existing data or assumptions about how the observations are generated. Therefore outliers are observations that deviate significantly from the expectations. Normal and the expected data observations are called inliers. From the viewpoint of information theory [1], outlier can be considered to carry interesting information. They consist of unusual, unexpected and new information in comparison to the normal operation data observations. Hawkins defines an outlier as *”an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism”* [2].

Outliers have varying names in different contexts and they can be generated by many different reasons. Names for outliers include fault [3], intrusion [4] and anomaly [5] and each name has a different meaning in their context. Outlier detection has been successfully applied in many applications in different fields. The concept of outlier detection has wide applicability in data analysis. Some of the recent applications from various fields include the following:

- Wireless network intruder detection [4].
- Enterprise application fault detection based on hardware performance data [5].
- Shipment optimization and fraud detection based on radio frequency identification data streams [6].
- Automatic segmentation of white matter hyperintensities in brain magnetic resonance images [7].

Outlier detection is a challenging task because the information for detecting outliers is extracted from the data itself in many real-world problems. Outlier detection is a context-dependent task because an outlier is a true outlier only in context and an outlier has a different interpretation in different environments.

1.1 Objectives and scope

The main objectives of this thesis are:

1. Specify a modular framework for outlier detection.
2. Develop an implementation of the framework.
3. Validate and verify the goodness of the ensemble method of three different outlier detection methods. The goodness is defined as a) accuracy and b) execution time:
 - (a) Capability to correctly detect outliers (accuracy).
 - (b) Execution time in seconds.
4. Compile ideas for future research work.

This thesis does not attempt to invent novel outlier detection algorithms, but studies the combination of existing ones. The outlier detection is limited to static (offline) datasets represented by a matrix \mathbf{X} . The analyzed data is completely contained inside a real matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ that has n rows and d columns of real numbers $x_{ij} \in \mathbb{R}$. The d columns are called variables, dimensions and matrix columns in this thesis. The n rows are called data observations and data points in this thesis. Vector $\mathbf{x}_i \in \mathbb{R}^{1 \times d}$ is a data point, which is a row in \mathbf{X} . The matrix \mathbf{X} consists of data points $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$. The framework is implemented using Python and C++. The framework combines three different and previously selected outlier detection algorithms.

1.2 Thesis structure

Chapter 2 surveys the related work in the field of combining outlier detection methods. The evaluated publications are chosen from the recent ensemble learning and outlier research papers. Chapter 3 introduces the preliminary structure of the framework and its modules. This chapter defines different outlier types, outlier algorithms and all other algorithms used in this thesis. Chapter 4 defines the modular framework for outlier detection. The framework definition is detached from the actual implementation because the

framework can be implemented using any object-oriented programming language. Chapter 5 presents the framework implementation used in this thesis. Chapter 6 presents the novelty and benefits of the framework. Chapter 7 describes and reports the experiments' results using public datasets and synthetic process data. Chapter 8 discusses the acquired results and contemplates future work ideas with the framework. Chapter 9 concludes this thesis by reviewing the previously mentioned objectives.

2 Related work

Outlier detection has been researched since the 19th century because it is important to find outliers in different application domains [8]. Chandola et al. stated that *”the importance of anomaly detection is due to the fact that anomalies in data translate to significant, and often critical, actionable information in a wide variety of application domains”* [8]. This makes outlier detection widely applicable in many domains where data is produced.

Multiple outlier detection methods can be combined as an outlier ensemble [9]. Ensemble learning is an approach that combines results of multiple models, such as outlier detection algorithms. The resulting model combination is called an ensemble and the combined models are called ensemble members. Many ensemble learning algorithms have been invented, including Adaptive Boosting (AdaBoost) [10] and Random Forests [11]. The use of ensembles is motivated by reduced prediction bias and variance [12, pp. 222] [13, pp. 653]. Bias is the prediction error of a model and variance is the expected error in unseen data. Small bias means that the model has learned data well because it can predict the data itself with small error. Small variance means that the model generalizes well between different data because it can predict different data with small error. Unfortunately a low bias causes a high variance and vice versa. This problem is called the bias-variance dilemma [12, pp. 114] or bias-variance trade-off [13, pp. 147].

The ensemble members of an outlier ensemble predict whether data point x is an outlier. The first formalization of outlier ensemble [14] was defined by Lazarevic et. al [15]. The following paragraphs introduce some of the recent outlier ensemble methods, which use ensembles for outlier detection.

Huang et al. [16] specified an ensemble for outlier detection called VoteOut. The algorithm combines two outlier detection methods. The first method is based on similar coefficient sum (SCS), which measures the deviation between data points [17]. Let r_{ij}

be called similarity coefficient between data points \mathbf{x}_i and \mathbf{x}_j . The r_{ij} is defined as

$$r_{ij} = 1 - \sqrt{\frac{1}{n} \sum_{k=1}^d (x_{ik} - x_{jk})^2}, \quad (1)$$

where n is the number of samples, d number of dimensions and x_{ij} is a real number in matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$.

After calculating every pairwise r_{ij} , a sum of similarity coefficients p_i is calculated for every data point. The p_i is defined as $p_i = \sum_{j=1}^n r_{ij}$. The final SCS_i for data point \mathbf{x}_i is calculated as $SCS_i = \frac{p_{max} - p_i}{p_{max}} * 100$, where p_{max} is the maximum p_i . The second method is based on kernel density estimation (KDE), which approximates a probability density function of a probability distribution [13, pp. 122]. Probability density function describes the relative likelihood of a given observation. Probability distribution assigns probabilities for possible observations. Probability density function specifies a probability distribution for continuous values. Let h denote a KDE parameter called bandwidth, \mathbf{x}_i row i in \mathbf{X} , n (d) the number of rows (columns) in \mathbf{X} and $K(\mathbf{x})$ a kernel function. Typical $K(\mathbf{x})$ for KDE is a Gaussian kernel $K(\mathbf{x}) = \frac{1}{\sqrt{2\pi}} e^{-\frac{\|\mathbf{x}\|^2}{2}}$ where $\|\mathbf{x}\|$ is the norm of \mathbf{x} [13, pp. 123] [12, pp. 51]. The h adjusts how sensitive the estimation is to data points in \mathbf{X} . Kernel density estimator $\hat{f}_h(\mathbf{x})$, which approximates the probability density of an arbitrary observation $\mathbf{x} \in \mathbb{R}^{1 \times d}$ [13, pp. 123] [12, pp. 51], can be defined as

$$\hat{f}_h(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^n K\left(\frac{\mathbf{x}_i - \mathbf{x}}{h}\right). \quad (2)$$

The results of the two integrated methods are combined as the final detection result. The research paper [16] reported better outlier and inlier detection rates for the ensemble, compared to the other separately tested outlier detection methods.

Schubert et al. [18] researched combining different outlier algorithms. The researchers stated that the resulting errors from the individual algorithms that are combined should be uncorrelated because the combination of similar results does not provide greater diversity in the results. Two variables are correlated if their values depend on each other. The

utilization of multiple uncorrelated methods would provide diverse results in the outlier detection because they provide a maximum amount of new information for the outlier detection. The researchers defined a framework for evaluating and comparing the accuracy of separate outlier detection methods. Schubert et al. devised an algorithm for selecting a subset of available outlier detection methods that maximize the diversity of results. The maximization of the diversity is based on the correlation. Experiments showed that combining outlier methods with uncorrelated errors increases the outlier detection accuracy.

Bouguessa [19] proposed a probabilistic approach for combining multiple outlier algorithms. The combined algorithms are required to return a degree of a data point being an outlier. The degree values are combined into a score vector for every data observation. The score vectors are modelled using a mixture model of beta distribution components. Mixture models are combinations of multiple components [20, pp. 432]. Typical components are probability distributions. Beta distribution is a probability distribution that is defined for random variable values $0 \leq x \leq 1$ [20, pp. 173]. Expectation-Maximization algorithm (EM) estimates the component parameters of the mixture model. EM is a method for finding the most likely parameter values of a mixture model and its components given the observed data [12, pp. 45]. The number of mixture components is inferred by minimizing Bayesian Information Criterion (BIC). BIC is used to select models by finding a trade-off between model complexity and ability to describe data [12, pp. 310]. BIC penalizes a high number of available parameters because it indicates a complex model. This is important because a complex model can lose the ability to generalize information. Average scores are calculated for every beta distribution component from their corresponding score vectors. The mixture component that has the highest average score is selected as the outlier component. Every data observation that originates from the outlier component is flagged as an outlier. The reported results were good compared to Local Outlier Factor (LOF, [21]), Local Density-based Outlier Factor (LDOF, [22]), k-Nearest Neighbor based outlier detection (k NN, [23]) and a weighted version of k NN (wk NN, [24]), which were used separately.

Nguyen et al. [25] proposed an ensemble framework (HeDES) for finding outliers in ran-

dom subspaces of a dataset. A random subspace is a random subset of the original dataset employing a random subspace of the available feature space. The feature space is a vector space defined by the used features which represent the properties of the inspected phenomenon. The d variables in dataset \mathbf{X} are the features of this phenomenon. Feature vectors consist of single features and they are d -dimensional points in the feature space. The method couples an approach called boosting [10] and random subspaces [11] for handling data without a priori knowledge of it. HeDES creates a synthetic dataset based on the unlabeled dataset and injects artificial outliers in the data. The artificial outliers are sampled from a uniform distribution that is bounded 10% beyond the observed maximum and minimum of variable values. Uniform probability distribution assigns equal probability for all observations between the two limits. HeDES assumes that a dataset of inliers is available because HeDES creates the injected outliers by modifying the available inliers. The modified dataset is used as training data in the random subspaces. The resulting model decides whether a data observation is an outlier or an inlier. HeDES experiments were reported to perform well with real-world datasets from UCI [26] machine learning repository. The next chapter defines different outlier types, outlier algorithms and different methods used in this thesis.

3 Preliminaries and methods for outlier detection

3.1 Outlier types

Outliers can be categorized into different types, which describe the nature of the outliers. This thesis divides outliers into two distinctive categories defined by Chandola et al [8]. The categories are point anomalies and contextual anomalies. The following sections introduce the outlier types.

3.1.1 Point anomalies

Point anomalies are anomalous data observations compared to the rest of the data [8]. The data itself is assumed to contain the information for correct outlier detection without having prior knowledge of the data [27]. Point anomalies are global outliers because they are found with respect to the entire data. Point anomalies do not consider how the observations are created and they respond to extreme deviations in the data. Point anomalies can be found by using thresholds, which are based on standard deviations and a mean value. Mean value $\bar{\mathbf{x}}$, standard deviation σ and variance σ^2 of vector \mathbf{x} with n elements are defined as

$$\begin{aligned}\bar{\mathbf{x}} &= \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i, \\ \sigma(\mathbf{x}) &= \sqrt{\frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})^2}, \\ \sigma(\mathbf{x})^2 &= \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})^2.\end{aligned}\tag{3}$$

Figure 1 illustrates one-dimensional data with two point anomalies, which are extreme values compared to the expected value. The dashed lines are thresholds that are calculated from mean and standard deviations.

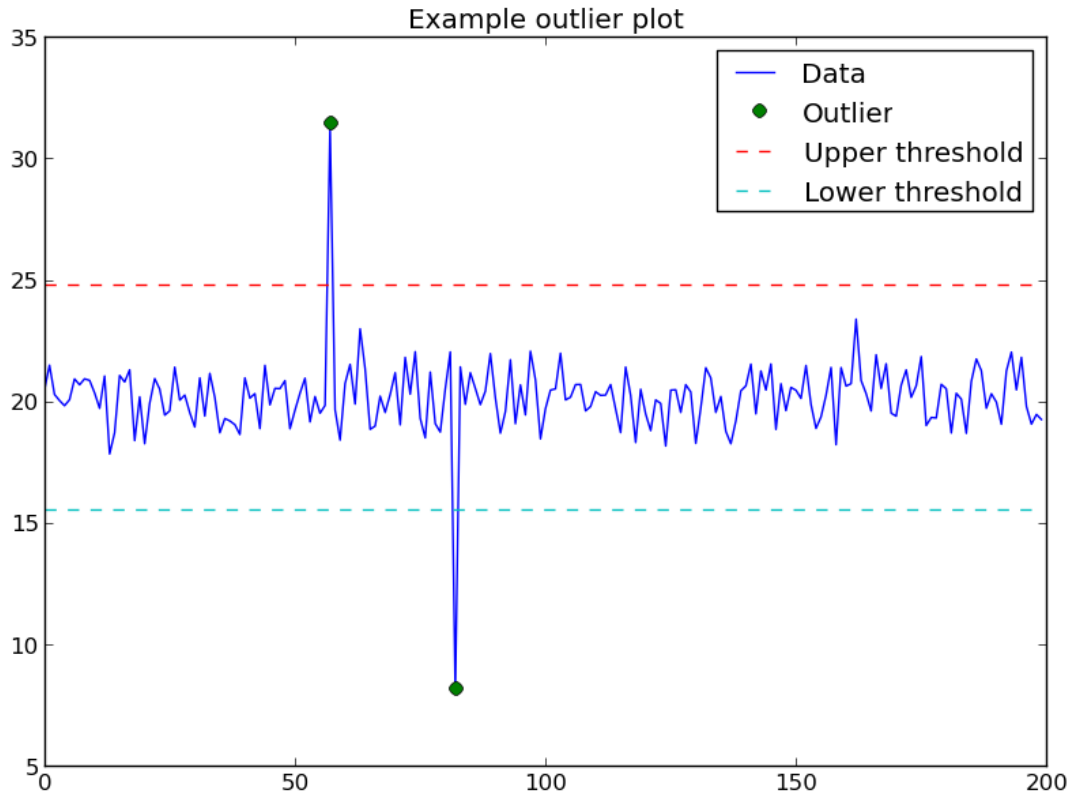


Figure 1: An example of time series data with two point anomalies. The outliers are marked with green dots. The outliers are extreme deviations from the expected values. The dashed lines indicate automatically calculated upper and lower thresholds. The thresholds are calculated as three times the standard deviation from the mean. One can find the two outliers without knowing how the time series data was created.

The data in the Figure 1 has the mean $\bar{x} = 20.17$ and standard deviation $\sigma(\mathbf{x}) = 1.54$. The two outliers have values 31.29 and 8.03. The data is normally distributed, which is defined using a probability density function $f(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$. The dashed lines are thresholds for determining if a data point is an outlier. The thresholds are calculated as three times the standard deviation from the mean. There is 99.73% probability of observing normally distributed values within the three standard deviations. This means that it is unlikely to observe values outside the range of the three standard deviations. The upper threshold is $\bar{x} + 3 * \sigma(\mathbf{x}) = 24.79$ and the lower threshold is $\bar{x} - 3 * \sigma(\mathbf{x}) = 15.55$. This use of the threshold successfully detects the two point anomalies because $31.29 > 24.79$ and $8.03 < 15.55$.

3.1.2 Contextual anomalies

Chandola et al. define contextual anomalies as data observations that are outliers only in a specific context [8]. Contextual attributes are measurable properties that define a context. The location of data in a time series is an example of a contextual attribute and it defines a context in a time series [8]. Algorithms for contextual anomalies use contextual attributes unlike point anomaly algorithms, which only compare the values of a single variable to its all available values or to statistics that are derived from the values. Contextual anomaly algorithms also consider how multiple variables are related to each other. Figure 2 is an example time series data that has contextual anomalies.

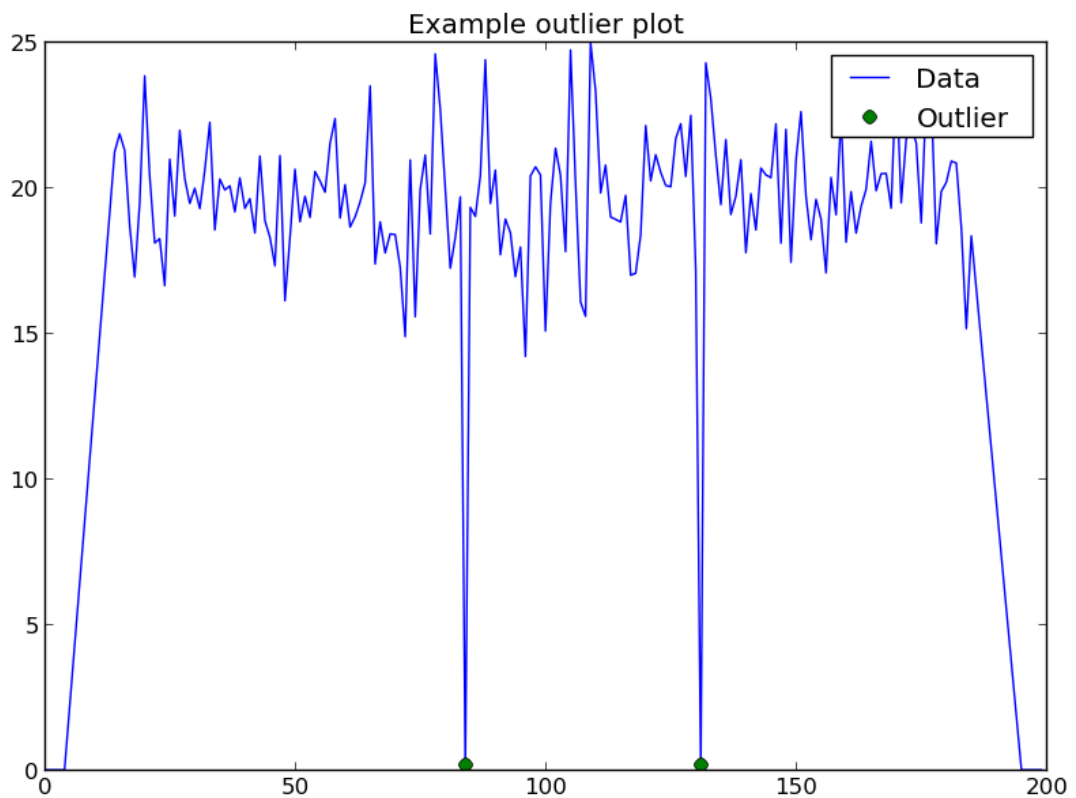


Figure 2: An example of time series data that has contextual anomalies (green dots). The outliers can not be discriminated from inliers by thresholding the values. Zero values in the beginning and end are inliers, but the zero values in the middle are outliers.

Zero values at beginning and end in Figure 2 are inliers. The two zero values in the middle are outliers. The outliers are contextual anomalies because the zero values are outliers if they have a specific location in the time series data. Point anomaly algorithms would threshold all zero values as outliers (see Figure 1), which is an incorrect classification. Mahalanobis distance [28] is an example method of finding contextual anomalies when data has multiple variables $d > 1$. Variable \mathbf{x}_i is defined as the i :th column in matrix \mathbf{X} . Mahalanobis distance uses information available in the inverse of a covariance matrix Σ to scale the variable values. Scaling means that the variable values are converted within a specified range of values. Inverse \mathbf{X}^{-1} of a matrix \mathbf{X} is defined as $\mathbf{X}^{-1}\mathbf{X} = \mathbf{X}\mathbf{X}^{-1} = \mathbf{I}$ where \mathbf{I} is an identity matrix. An identity matrix \mathbf{I} is a square matrix (n columns and rows) in which the main diagonal has the values of one and the other elements have the value of zero. The inverse matrix of an identity matrix \mathbf{I}^{-1} is an identity matrix $\mathbf{I}^{-1} = \mathbf{I}$. The covariance matrix Σ of matrix \mathbf{X} is defined as

$$\Sigma = \mathbf{E}[(\mathbf{X} - \mathbf{E}[\mathbf{X}])(\mathbf{X} - \mathbf{E}[\mathbf{X}])^T], \quad (4)$$

where the superscript symbol T denotes the transpose matrix \mathbf{X}^T of \mathbf{X} and the $\mathbf{E}[\mathbf{x}]$ is the expected value of variable \mathbf{x} . The transpose \mathbf{X}^T of matrix \mathbf{X} transforms its rows into columns, which defined as $\mathbf{X}_{ij} = (\mathbf{X}^T)_{ji}$ for all i and j .

The covariance matrix elements Σ_{ij} are covariances between variables \mathbf{x}_i and \mathbf{x}_j . The covariance between two variables measures how much they change together. The covariance of a variable with itself $\Sigma_{ii} = \sigma^2(\mathbf{x}_i)$ is its variance σ^2 . Mahalanobis distance is defined between points \mathbf{x}_1 and \mathbf{x}_2 with covariance matrix Σ [29, pp. 19–20] [12, pp. 30]:

$$D(\mathbf{x}_1, \mathbf{x}_2, \Sigma) = \sqrt{(\mathbf{x}_1 - \mathbf{x}_2)^T \Sigma^{-1} (\mathbf{x}_1 - \mathbf{x}_2)}. \quad (5)$$

Unlike point anomalies, the contextual outliers can consider the underlying correlation structure of variables. The covariance matrix explains the correlations between variables and forms the correlation structure. Mahalanobis distance $D(\mathbf{x}_1, \mathbf{x}_2, \mathbf{I})$ that uses an identity matrix \mathbf{I} as the covariance matrix reduces to $\sqrt{(\mathbf{x}_1 - \mathbf{x}_2)^T (\mathbf{x}_1 - \mathbf{x}_2)}$. This is evident as $\mathbf{I}^{-1} = \mathbf{I}$ and $\mathbf{X}\mathbf{I} = \mathbf{X}$. Mahalanobis distance $D(\mathbf{x}_1, \mathbf{x}_2, \mathbf{I})$ is known as the Euclidean

distance $D(\mathbf{x}_1, \mathbf{x}_2)$, which is defined between points \mathbf{x}_1 and \mathbf{x}_2 [29, pp. 80]:

$$D(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{(\mathbf{x}_1 - \mathbf{x}_2)^T (\mathbf{x}_1 - \mathbf{x}_2)}. \quad (6)$$

3.2 k-Means clustering

Clustering algorithms group similar data points together. The groups are called clusters. The similarity is a quantified measure, which is calculated using a function. The similarity measure can be a distance, which represents the proximity of two data points. Mahalanobis distance (Eq. 5) and Euclidean distance (Eq. 6) are examples of the available distance functions. Every data point $\mathbf{x}_i \in \mathbf{X}$ is grouped into a cluster $j \in \{1, 2, \dots, k\}$ where j denotes the cluster index. [12, pp. 600–603]

k-Means clustering algorithm is an iterative clustering algorithm. Iterative algorithms generate approximate solutions to problems and they are executed in a sequence of steps. k-Means has only one parameter k , which is the number of clusters. Every cluster has a centroid $\mathbf{c}_j = \bar{J}$ where $J \subset \mathbf{X}$ is a data subset that belongs to cluster j . The cluster j centroid \mathbf{c}_j is the mean value of data points J in the cluster. Cluster centroid \mathbf{c}_j is defined as

$$\mathbf{c}_j = \frac{1}{n} \sum_{\mathbf{x}_i \in J} \mathbf{x}_i \quad (7)$$

where n is the number of data points in J [12, pp. 617,741–742]. k-Means assigns every data point \mathbf{x}_i to the closest cluster measured with the distance between the data point and a cluster centroid. The closest cluster has the smallest distance measured to a data point. Euclidean distance is a typical choice for the distance function. The centroids \mathbf{c}_j have to be initialized first. A commonly used approach is to select random data points $\mathbf{x}_i \in \mathbf{X}$ as the initial centroids. After every data point is assigned into cluster $j \in \{1, 2, \dots, k\}$, the cluster centroids are calculated (Eq. 7). The cluster assignments and cluster centroids are iteratively updated until the cluster centroids do not change. k-Means clustering is

explained in Algorithm 1, which describes how the k-means clustering works. [29, pp. 424–425]

Algorithm 1 k-Means algorithm pseudocode.

- 1: **Require:** Dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$
 - 2: Select random data points \mathbf{x}_i as the initial centroids \mathbf{c}_j
 - 3: Assign every data point \mathbf{x}_i into a cluster $j \in \{1, 2, \dots, k\}$
 - 4: **while** Cluster centroids change **do**
 - 5: Calculate new cluster centroids \mathbf{c}_j
 - 6: Assign the data points to the closest cluster centroids
 - 7: **end while**
-

Figure 3 shows an example of scatter plots of k-means clustering results and how the clustering is affected when the parameter k is adjusted. The test data is artificially generated to contain three distinctive clusters. The plot colors indicate the final cluster for any given data point. Different colors depict the different clusters. The initial selection of parameter k is critical for successful clustering. Figure 3 shows that $k = 2$ is too small because distant points are clustered together inside the blue cluster. Parameter $k = 4$ value is too big because it forces one cluster to be divided. Optimal clustering is achieved with $k = 3$ because the three distinctive groups in the original data become clusters.

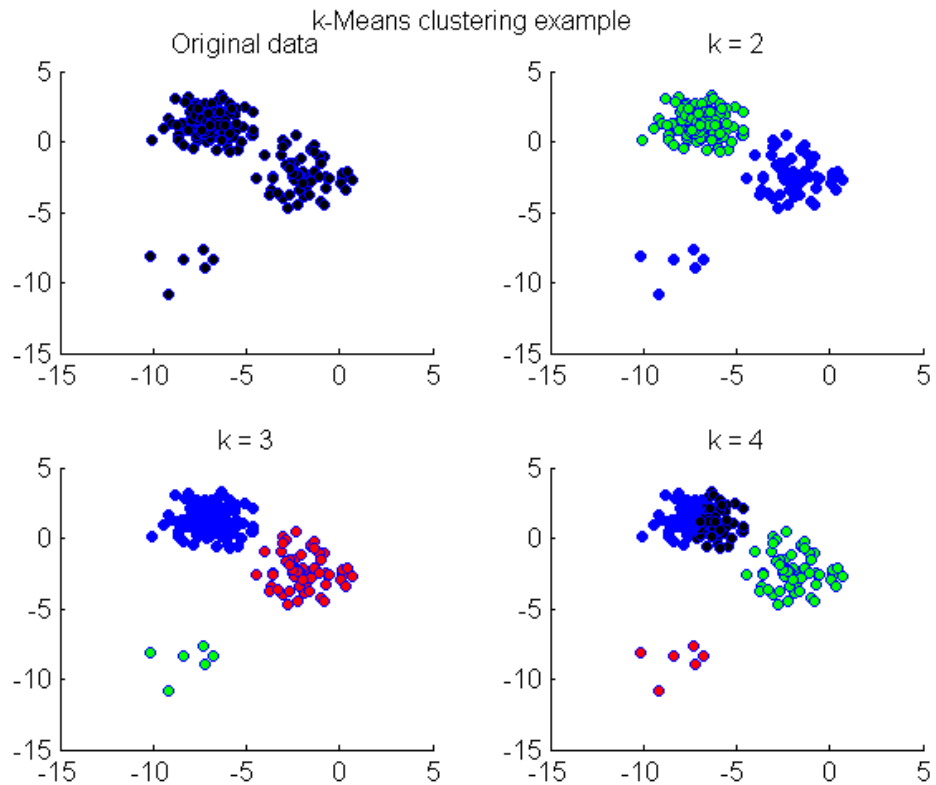


Figure 3: An example of k-means clustering. The original data has three distinctive groups of data points. Parameter $k = 2$ has too few clusters and $k = 4$ has too many clusters. Value $k = 3$ results in optimal clustering where the three groups are clustered separately.

3.3 Outlier detection algorithms

Outlier detection algorithms attempt to detect outliers in dataset \mathbf{X} . The algorithms define normal data as a region in the feature space, which is called a normal region. Outlier is a data observation $\mathbf{x}_i \in \mathbf{X}$ that does not belong in the normal region. Determining the normal region is a challenging task [8] because of the following:

1. Data and outlier types are specific to application domain.
2. The normal region of the feature space may change as time passes.
3. Typically the outliers in a dataset \mathbf{X} are not known in advance.

The application domain imposes requirements for the outlier detection. An outlier algorithm is selected to detect outliers of a specific type (Chapter 3.1), which are present in the analyzed data. The application domain governs the type of the present outliers. The application domain also specifies the feature space, which governs the type of available data. The data can have an arbitrary number of features d and data observations n . The d features can contain continuous values, discrete values or a collection of continuous and discrete values. This restricts the number of suitable algorithms for outlier detection. [8]

Outlier detection is related to noise removal, novelty detection and statistical process monitoring (SPM) [8]. Noise is uninteresting and unwanted information in data. Novelty is a previously unseen data pattern in a new data observation. Outliers are not noise or novelties because Chandola et al. state that outliers are *”patterns in data that do not conform to a well defined notion of normal behavior”* [8]. SPM attempts to detect faults in an industrial process, which is a phenomenon that produces a continuous transmission of sensor values. SPM also typically attempts to determine the cause of an outlier detection. SPM is explained in detail in Chapter 3.5.

There is no established taxonomy for outlier algorithms. There are some recurring notions seen in outlier publications, including a division into global and local methods [9, 25, 30]. Global methods compare data observations against all available data and find outliers with respect to the entire dataset. Local methods examine local neighborhoods of data observations. The local neighborhood of a data observation is a k -neighborhood, which is defined as the k nearest data points for a data point \mathbf{x}_i . The distance is measured using a distance function D . The resulting set of k nearest data points for \mathbf{x}_i is denoted as $\mathcal{N}(\mathbf{x}_i, k)$. The notation was used by Zhang et al. [22]. Data point \mathbf{x}_i is called a local neighbor of data point \mathbf{x}_j if \mathbf{x}_i is in the k -neighborhood $\mathcal{N}(\mathbf{x}_j, k)$ of \mathbf{x}_j . Figure 4 is an example of finding 6 nearest neighbors in two-dimensional data. The yellow dots are the $\mathcal{N}(\mathbf{x}_i, 6)$ for data point \mathbf{x}_i , which is the red dot. The blue ellipse denotes the area within the 6-neighborhood.

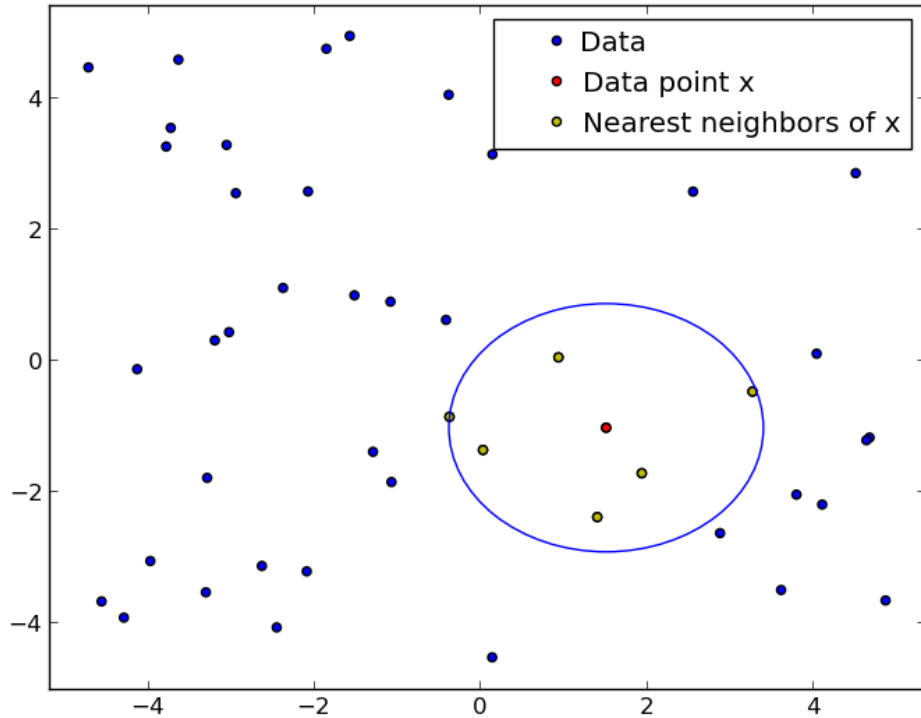


Figure 4: Example of finding 6 nearest neighbors $\mathcal{N}(x, 6)$ for the red dot x . The resulting 6-neighborhood is denoted by the blue ellipse.

Some outlier algorithms [21, 22, 31, 32] use a quantified measure to indicate the likeliness of a data point x_i being an outlier. This value is called a score. A higher score implies a more likely outlier detection. The score does not have a maximum value. This means that it is hard to define the magnitude of a high score, which is used to detect outliers [33]. An outlier is a data point x_i that has the corresponding score above a threshold value T . This threshold value is selected in advance by an analyst [8]. Scores that are calculated by outlier algorithms vary in their scale and range [9]. Two different algorithms may calculate a different score for the same data point x_i . Unfortunately even subsets of same data can result in different scores [30]. A single algorithm may calculate varying scores for the same data point x_i if different subsets of data are used. Therefore it is very hard to compare the scores between different algorithms and datasets [9]. Selecting a universal and unambiguous threshold T is also very hard because the scores are very unstable [30, 33].

Outlier detection algorithms can be also categorized by how they detect outliers. The following list categorizes outlier detection algorithms based on how they detect outliers. Many outlier publications [18, 21, 22, 32] define similar categories.

1. Distribution-based outlier detection [34, 35]

The normal region in feature space is modeled as a region of high probability in a probability distribution. Data observations with a low probability in the probability distribution are assumed to be outliers. The probability distribution is required to correctly represent the data. This is very important because the data is assumed to originate from the used probability distribution.

2. Distance-based outlier detection [21, 22]

The outlier status of a data observation is determined with its distance to other data observations. The distances are calculated with a distance function D . Data observations that have a high distance to other data observations are outliers.

3. Density-based outlier detection [21, 32]

Outliers are data observations, which are located in regions with low density $p(\mathbf{x})$ in feature space. Density $p(\mathbf{x})$ is the proportion of data points within a volume V . The density $p(\mathbf{x})$ is approximated as

$$p(\mathbf{x}) \simeq \frac{k}{nV}, \quad (8)$$

where n is the number of all samples and k is the number of samples inside volume V .

4. Clustering-based outlier detection [31, 36]

Clustering is not designed for outlier detection because the goal of clustering is to group similar data. This means that clustering has to be applied as a step in cluster-based outlier detection. Clustering-based outlier methods commence outlier detection by clustering dataset \mathbf{X} . Outliers are detected using the following approaches.

- A data observation \mathbf{x}_i with large distance $D(\mathbf{x}_i, \mathbf{c}_j)$ to the closest cluster centroid \mathbf{c}_j is an outlier.
- The clusters with low density and low number of members \mathbf{x}_i are outliers clusters. The data observations \mathbf{x}_i in the outlier clusters are outliers.
- Not every clustering algorithm [37] requires all data to be assigned in clusters. The data observations that are not assigned in clusters are outliers [8].

Figure 5 illustrates and summarizes the outlier method taxonomy that is used in this thesis.

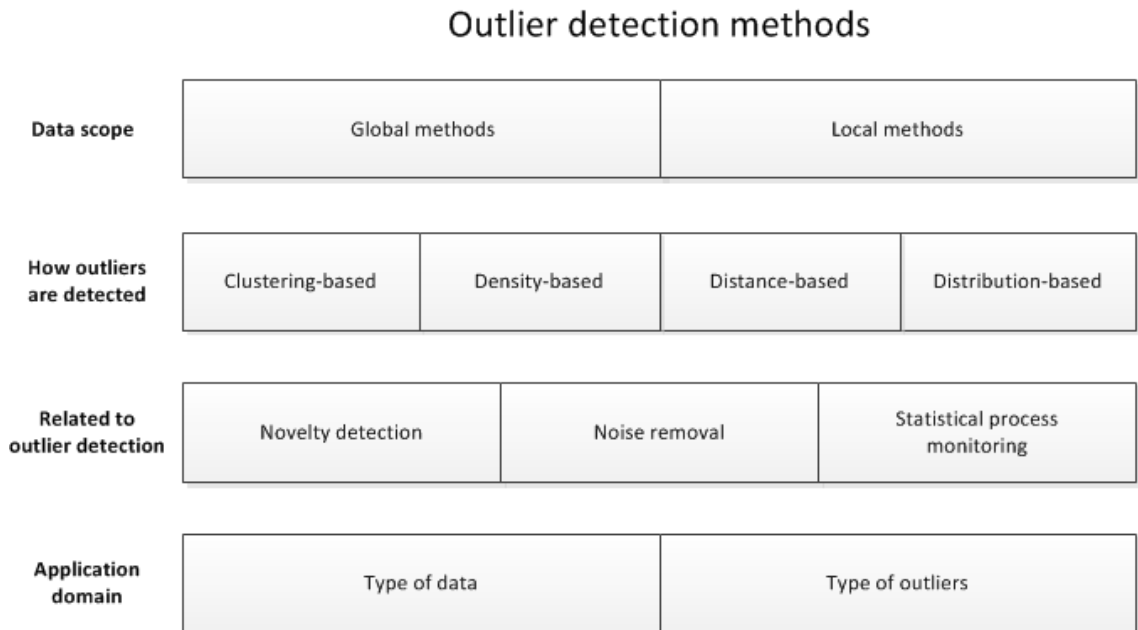


Figure 5: The taxonomy of outlier detection methods that is used in this thesis.

This thesis combines the following three outlier methods:

- Local Distance-based Outlier Factor (LDOF) [22]

LDOF is a distance-based outlier detection method. Outliers are data points with a

high distance to their nearest data points compared to the average distances among the nearest data points. Chapter 3.3.2 introduces the LDOF in detail.

- Cluster-based Outlier Detection (CBOD) [31]

CBOD is a clustering-based outlier detection method, which finds outliers by determining outlier clusters. Chapter 3.3.3 introduces the CBOD in detail.

- Statistical Process Monitoring (SPM) [38–40]

SPM is a quality control method for finding outliers in data with multiple variables ($d > 1$). Chapter 3.5 introduces the SPM in detail.

LDOF, CBOD and SPM were selected because each one has a different approach for detecting outliers. LDOF is a local method based on distances, CBOD is a global method based on clustering and SPM is a quality control method, which is a global method. LOF and LDOF are local methods because they only consider the k -neighborhood. Data points that are not in $\mathcal{N}(\mathbf{x}_i, k)$ are ignored. SPM and CBOD are global methods because they find outliers by using all data points \mathbf{x}_i of a dataset \mathbf{X} . This thesis combines LDOF, CBOD and SPM. The high diversity between the three algorithms demonstrates the idea that arbitrary outlier detection algorithms can be combined. Local Outlier Factor (LOF) [21] was selected to provide comparable results because it forms the baseline in many outlier publications [21,22,30,32,33]. LOF is a density-based outlier detection method, which is different from LDOF and CBOD. LOF was used separately. LOF, LDOF and CBOD use a score to indicate the likeliness of an outlier detection. Table 1 summarizes the outlier methods that are used in this thesis. The following sections introduce the used outlier detection algorithms in detail.

Table 1: The outlier methods that are used in this thesis. The methods have different approaches for detecting outliers. This work combines LDOF, CBOD and SPM for outlier detection. LOF provides comparable results for evaluation.

Outlier detection method	Category	Uses score	Data scope
LDOF	Distance-based	Yes	Local
CBOD	Clustering-based	Yes	Global
SPM	-	No	Global
LOF	Density-based	Yes	Local

3.3.1 Local Outlier Factor

Local Outlier Factor (LOF) is a well-established outlier detection algorithm [21]. LOF is based on density $p(\mathbf{x})$ where outliers are located in low density neighborhoods. Let $D_k^{\mathbf{x}_j}$ be the distance of \mathbf{x}_j to its k :th nearest neighbor in $\mathcal{N}(\mathbf{x}_j, k)$ and $D(\mathbf{x}_i, \mathbf{x}_j)$ the distance from \mathbf{x}_i to \mathbf{x}_j . The $D_k^{\mathbf{x}_j}$ is used to calculate reachability distance $reachdist_k(\mathbf{x}_i, \mathbf{x}_j)$ between points \mathbf{x}_i and \mathbf{x}_j . The reachability distance is the maximum of $D_k^{\mathbf{x}_j}$ and $D(\mathbf{x}_i, \mathbf{x}_j)$ in which the greater of the two values is chosen. The reachability distance is at least the distance of $D_k^{\mathbf{x}_j}$ and it is bigger than the distance of $D_k^{\mathbf{x}_j}$ if \mathbf{x}_i is not a local neighbor of \mathbf{x}_j . The reachability distance is defined as

$$reachdist_k(\mathbf{x}_i, \mathbf{x}_j) = \max(D_k^{\mathbf{x}_j}, D(\mathbf{x}_i, \mathbf{x}_j)). \quad (9)$$

Let $avgreach(\mathbf{x}_i)$ be the average reachability distance between \mathbf{x}_i and all the data points \mathbf{x}_j in the k -neighborhood $\mathbf{x}_j \in \mathcal{N}(\mathbf{x}_i, k)$ of \mathbf{x}_i . The $avgreach(\mathbf{x}_i)$ is defined as

$$avgreach(\mathbf{x}_i) = \frac{\sum_{\mathbf{x}_j \in \mathcal{N}(\mathbf{x}_i, k)} reachdist_k(\mathbf{x}_i, \mathbf{x}_j)}{k}. \quad (10)$$

The $avgreach(\mathbf{x}_i)$ is used to calculate local reachability density $lrd_k(\mathbf{x}_i)$ for data point \mathbf{x}_i . The local reachability density is the inverse of $avgreach(\mathbf{x}_i)$. Local reachability density is defined as

$$lrd_k(\mathbf{x}_i) = \frac{1}{avgreach(\mathbf{x}_i)} = \frac{k}{\sum_{\mathbf{x}_j \in \mathcal{N}(\mathbf{x}_i, k)} reachdist_k(\mathbf{x}_i, \mathbf{x}_j)}. \quad (11)$$

The final LOF score, $LOF_k(\mathbf{x}_i)$, is based on $reachdist_k(\mathbf{x}_i, \mathbf{x}_j)$, $avgreach(\mathbf{x}_i)$ and $lrd_k(\mathbf{x}_i)$. The LOF score $LOF_k(\mathbf{x}_i)$ for a data point \mathbf{x}_i is based on the average lrd of neighborhood $\mathcal{N}(\mathbf{x}_i, k)$ divided by the $lrd_k(\mathbf{x}_i)$ as follows:

$$LOF_k(\mathbf{x}_i) = \frac{\sum_{\mathbf{x}_j \in \mathcal{N}(\mathbf{x}_i, k)} lrd_k(\mathbf{x}_j)}{lrd_k(\mathbf{x}_i) * k}. \quad (12)$$

The LOF_k can be interpreted as a degree of measure how packed a given data observation is in a locally reachable neighborhood [22], where no assumptions are made of the distribution of all data. High LOF_k score means that \mathbf{x}_i has a deviating density compared to its neighborhood, which indicates that \mathbf{x}_i is an outlier.

3.3.2 Local Distance-based Outlier Factor

Local Distance-based Outlier Factor (LDOF) is a distance-based outlier detection algorithm [22]. LDOF detects outliers that are distant to groups of neighboring data points. LDOF aims to find outliers in datasets where the number of dimensions d is high. LDOF is different than LOF because LDOF relies on distances instead of densities. Let $\bar{D}_k^1(\mathbf{x}_i)$ be the average distance of \mathbf{x}_i to its k neighbors in $\mathcal{N}(\mathbf{x}_i, k)$. The distance is measured using a distance function D . The average distance is defined as

$$\bar{D}_k^1(\mathbf{x}_i) = \frac{1}{k} \sum_{\mathbf{x}_j \in \mathcal{N}(\mathbf{x}_i, k)} D(\mathbf{x}_i, \mathbf{x}_j). \quad (13)$$

LDOF also calculates $\bar{D}_k^2(\mathbf{x}_i)$, which is the average distance among the data points in $\mathcal{N}(\mathbf{x}_i, k)$. The $\bar{D}_k^2(\mathbf{x}_i)$ is defined as

$$\bar{D}_k^2(\mathbf{x}_i) = \frac{1}{k(k-1)} \sum_{\mathbf{x}_j, \mathbf{x}_k \in \mathcal{N}(\mathbf{x}_i, k), i \neq j} D(\mathbf{x}_j, \mathbf{x}_k). \quad (14)$$

LDOF calculates a score $LDOF_k(\mathbf{x}_i)$, which is a measure of a data point \mathbf{x}_i being an outlier in its neighbourhood. The value is calculated as the average distance of \mathbf{x}_i to its neighbors divided by the average distance among the data points in $\mathcal{N}(\mathbf{x}_i, k)$. The $LDOF_k$ measures how isolated \mathbf{x}_i is from its neighbors $\mathcal{N}(\mathbf{x}_i, k)$. The LDOF score is

defined as

$$LDOF_k(\mathbf{x}_i) = \frac{\bar{D}_k^1(\mathbf{x}_i)}{\bar{D}_k^2(\mathbf{x}_i)}. \quad (15)$$

LDOF is shown [22] to have a threshold 0.5 when $k, n \rightarrow \infty$ and $\bar{D}_k^2(\mathbf{x}_i) \rightarrow 0$. The threshold classifies the most of the outliers with a high probability. This means that theoretically the most of the outliers are data points \mathbf{x}_i with $LDOF_k(\mathbf{x}_i) > 0.5$.

3.3.3 Clustering-based Outlier Detection

Outlier detection that uses clustering detects outliers by determining which clusters deviate significantly from other clusters. This is a fundamentally different concept than distance-based (LDOF) or density-based (LOF) outlier detection. Clusters that deviate enough from other clusters are assumed to contain outlier data. Clusters with outliers (inliers) are called outlier (inlier) clusters.

Many clustering-based outlier detection methods [31,41–44] have been introduced. Clustering-based Outlier Detection (CBOD) is an algorithm [31] for finding outliers in dataset \mathbf{X} . The algorithm clusters the dataset using a clustering algorithm. This thesis uses k-means clustering to cluster the dataset \mathbf{X} in the k clusters $C = \{C_1, C_2, \dots, C_k\}$, where every cluster C_i contains a subset of \mathbf{X} . The number of members in cluster C_i is denoted by $|C_i|$.

CBOD calculates Outlier Factor (OF) for every cluster, which measures the likeliness of cluster C_i being an outlier cluster. The OF of cluster C_i is a weighted sum of distances to other $k - 1$ clusters and their number of cluster members $|C_j| \forall i (i \neq j)$. The distance is measured using a distance function D . Higher OF value implies a more likely outlier cluster. Outlier Factor of a cluster C_i is as

$$OF(C_i) = \sum_{j \neq i} |C_j| * D(C_i, C_j), \quad (16)$$

where $|C_j|$ is the number of members in cluster C_j .

CBOD uses a parameter $\epsilon \in [0, 1]$ to approximate the ratio of outliers to the number of samples n in dataset \mathbf{X} . The ratio is not typically known in advance. Jiang et al. [31] suggest to select ϵ in the range of $\epsilon \in [0.05, 0.1]$ if no a priori knowledge is available of dataset \mathbf{X} . Assume that the value of parameter ϵ is selected in advance ($\epsilon \in [0, 1]$) and the clusters $C = \{C_1, C_2, \dots, C_k\}$ are sorted by their OF values $OF(C_1) \geq OF(C_2) \geq \dots \geq OF(C_k)$. One can attempt to determine outlier clusters by selecting b clusters with the highest OF values until $\epsilon * n$ observations are selected. Clusters C_1, C_2, \dots, C_b are considered as outlier clusters and the corresponding data points are outliers. The method is mathematically defined as finding a minimum value for b as

$$\frac{\sum_{i=1}^b |C_i|}{n} \geq \epsilon, \quad (17)$$

where n is the total number of data samples in \mathbf{X} .

3.4 Principal component analysis

Principal Component Analysis (PCA), also known as the Karhunen-Loève transform [12, pp. 326], is a well-known ordination method. Ordination methods describe data by grouping similar data points near each other. PCA is typically used for data dimensionality reduction [29, pp. 326] where resulting data uses l columns instead of d columns ($l \leq d$). Other potential uses of PCA include outlier detection.

Humans can not effectively comprehend the underlying data structure when the number of dimensions d is high. Data with a high number of dimensions has a massive number of different value combinations. Every added dimension grows the data volume exponentially. If a good data representation with $d = 1$ has n samples, then n^d samples are needed for the same density with $d \geq 1$ [12, pp. 55]. The exponential growth quickly results in sparse and insufficient data representation. The problem is called the curse of dimensionality. Dimensionality reduction is an important step in successful data analysis [20, pp. 329] [29, pp. 559] [12, pp. 323] [45].

Let matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ be a real valued matrix with n rows and d columns. PCA projects the original data $\mathbf{X} \in \mathbb{R}^{n \times d}$ to a $\mathbf{X} \in \mathbb{R}^{n \times l}$ subspace [29, pp. 559] spanned by principal components \mathbf{p}_i (PC) where $d \geq l \geq 1$. This subspace is called principal subspace. PCs $\mathbf{p}_i \in \mathbb{R}^d = \{p_1, p_2, \dots, p_d\}$ are d -dimensional vectors, which are represented as rows in a loadings matrix $\mathbf{P} \in \mathbb{R}^{l \times d} = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_l\}^T$. The loadings (matrix \mathbf{P} values) define a basis for projecting \mathbf{X} from d -space to l -space [46]. Let \mathbf{p}_{ij} be the j :th element p_j in PC \mathbf{p}_i . PCs define uncorrelated linear combinations $\mathbf{p}_{i1}x_1 + \mathbf{p}_{i2}x_2 + \dots + \mathbf{p}_{id}x_d$ for a data point $\mathbf{x}_i = \{x_1, x_2, \dots, x_d\}$ with the original d variables. The real valued score matrix $\mathbf{T} \in \mathbb{R}^{n \times l}$ represents \mathbf{X} in l -space, where every row of score matrix \mathbf{T} corresponds to a row in \mathbf{X} . The score matrix \mathbf{T} of \mathbf{X} in l -space is defined as

$$\mathbf{T} = \mathbf{P}^T \mathbf{X} , \quad (18)$$

where \mathbf{P} is a loadings matrix. The scores \mathbf{T} in l -space can be used for clustering or any other data analysis in place of the original data \mathbf{X} in d -space because \mathbf{T} is a lower dimensional representation of \mathbf{X} . Matrix \mathbf{E} is a residual matrix $\mathbf{E} = \mathbf{X} - \mathbf{TP}^T$ containing error residuals. The residuals are information that the PCA model does not explain with the l PCs. PCA models matrix \mathbf{X} as

$$\mathbf{X} = \mathbf{TP}^T + \mathbf{E} . \quad (19)$$

Matrix multiplication $\mathbf{X}\mathbf{x}$ between matrix \mathbf{X} and vector \mathbf{x} rotates the vector \mathbf{x} and changes its magnitude. Eigenvector v is a vector that denotes a direction, which is not affected by the rotation. Every eigenvector v has a corresponding eigenvalue λ . The eigenvalue λ denotes the magnitude of the invariant direction, which results after the matrix multiplication $\mathbf{X}v$. Eigenvector v and eigenvalue λ of a matrix \mathbf{X} are defined to satisfy the Equation $\mathbf{X}v = \lambda v$. Multiplication of \mathbf{X} and its eigenvector v results in a multiple λv of v . This means that the eigenvector preserves its direction in the multiplication $\mathbf{X}v$. Loadings are eigenvectors v of $\Sigma \in \mathbb{R}^{d \times d}$, which is the covariance matrix of dataset \mathbf{X} . The eigenvectors and eigenvalues can be calculated using matrix eigendecomposition that

is defined as

$$\mathbf{X} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}, \quad (20)$$

where matrix \mathbf{Q} contains the eigenvectors and $\mathbf{\Lambda}$ the eigenvalues of matrix \mathbf{X} . PCs \mathbf{p}_i project the original data to principal subspace with variance σ_i^2 . The PCs are said to explain the variance. The amount of explained variance σ_i^2 of \mathbf{p}_i is proportional to the eigenvalue λ_i element ($\sigma_i^2 \propto \lambda_i$). The eigenvalue λ_i is the diagonal element Λ_{ii} in matrix $\mathbf{\Lambda}$. The PCs are ordered in descending order by the explained variance $\sigma_1^2 \geq \sigma_2^2 \geq \sigma_3^2 \geq \dots \geq \sigma_l^2$. The first PC \mathbf{p}_1 explains the most of the original data variance.

Figure 6 illustrates an example of PCA, where PCA reduces Fisher's Iris dataset [47] from four dimensions into a two-dimensional feature space. Pattern recognition literature uses Fisher's Iris data set frequently, which contains 50 petal and sepal measurements of three related Iris flower species. The importance of the original variables for the resulting PCs is plotted using blue lines, and the red scatter points are original data rows projected to the two-dimensional feature space.

3.4.1 Singular value decomposition

Factorization is a process that reduces an object into multiple elements. The product of the elements returns the original factorized object. Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ denote a matrix with n rows and d columns. Singular value decomposition (SVD) factorizes matrix \mathbf{X} into matrices \mathbf{V} , \mathbf{U} and $\mathbf{\Sigma}$ [20, pp. 662]. The factorization is defined as

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T. \quad (21)$$

The columns of matrix \mathbf{V} are called left singular vectors of \mathbf{X} , which are the eigenvectors of $\mathbf{X}^T\mathbf{X}$. The columns of matrix \mathbf{U} are called right singular vectors of \mathbf{X} , which are the eigenvectors of $\mathbf{X}\mathbf{X}^T$. Matrices \mathbf{V} and \mathbf{U} are unitary matrices [12, pp. 335]. A unitary matrix is defined as a matrix with the following property: the inverse of a unitary matrix \mathbf{X} is its transpose $\mathbf{X}^{-1} = \mathbf{X}^T$. This guarantees that matrix multiplication $\mathbf{X}^T\mathbf{X}$ of a unitary

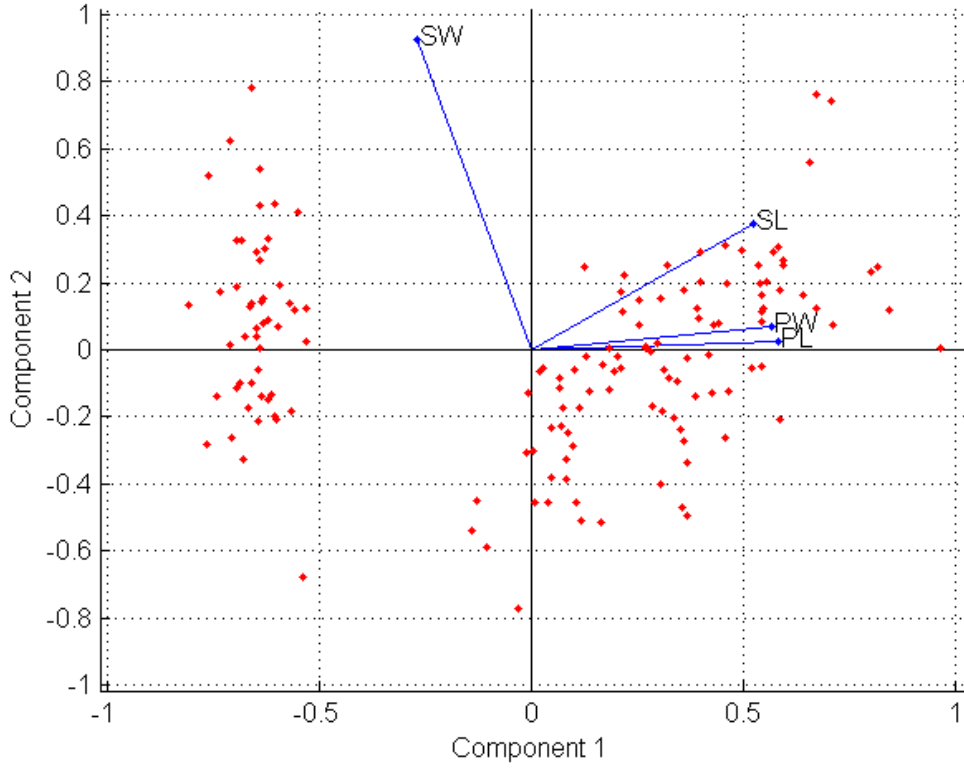


Figure 6: PCA example where the Fisher Iris dataset is projected to two-dimensional principal subspace. The blue lines are the original variables and the red scatter points are projected data points. The original variables are flower sepal and petal measurements: PW=petal width, PL=petal length, SW=sepal width and SL=sepal length.

matrix with its transpose results in identity matrix \mathbf{I} . The properties are defined as

$$\begin{aligned}\mathbf{X}^T \mathbf{X} &= \mathbf{X} \mathbf{X}^T = \mathbf{I}, \\ \mathbf{X}^{-1} &= \mathbf{X}^T.\end{aligned}\tag{22}$$

The diagonal matrix is defined as a matrix that has zero values outside its main diagonals. Matrix Σ is a diagonal matrix and contains singular values, which are defined as the square roots of the non-zero eigenvalues of $\mathbf{X} \mathbf{X}^T$ and $\mathbf{X}^T \mathbf{X}$. The singular values in Σ are sorted in descending order. If columns of the matrix \mathbf{X} have the mean value of zero ($\bar{x}_i = 0$), $\mathbf{X}^T \mathbf{X}$ is the covariance matrix of \mathbf{X} .

Matrix \mathbf{V} is the loading matrix \mathbf{P} for PCA, which is used for calculating the PC scores \mathbf{T} . SVD is typically used to implement PCA because SVD factorizes every matrix \mathbf{X} into $\mathbf{U}\Sigma\mathbf{V}^T$ and provides the loadings matrix \mathbf{P} . SVD has also a number of other useful applications including matrix rank determination [48], which is the maximum number of independent rows or columns in a matrix. Many SVD implementations are available (Eigen [49], SciPy [50]). MATLAB [51] also offers the SVD implementation. [12, pp. 335–337]

3.5 Statistical process monitoring

SPM monitors a process by analyzing univariate ($d = 1$) or multivariate ($d \geq 2$) data for outliers. The outliers are called faults in the SPM context. SPM is typically based on an ordination method like PCA [39]. SPM is used to detect faults in online sensor data streams, which are continuous transmissions of sensor values. The ordination model is created using sensor data that is fault free and recorded during normal operating conditions. SPM research has created a variety of different statistics to detect the faults. Typical statistics for detecting faults with PCA are Q -statistic and Hotelling's T^2 -statistic [39]. Q -statistic measures data variation in a subspace, which is spanned by the last $d - l$ unused principal components. This subspace is called residual subspace. The loadings in residual subspace are called residual loadings $\tilde{\mathbf{P}} \in \mathbb{R}^{(d-l) \times d}$, which are the last $d - l$ unused loadings. Hotelling's T^2 -statistic measures data variation in principal subspace using eigenvalues in matrix $\mathbf{\Lambda}$ and loadings matrix \mathbf{P} . The magnitude of the statistics Q and T^2 values indicate the degree of a data point \mathbf{x}_i being faulty. This is similar to the concept of the outlier score that is calculated by outlier algorithms. The statistics Q and T^2 for a data point \mathbf{x}_i are defined using matrices \mathbf{P} , $\tilde{\mathbf{P}}$ and $\mathbf{\Lambda}$ as follows:

$$Q = \mathbf{x}_i^T \tilde{\mathbf{P}} \tilde{\mathbf{P}}^T \mathbf{x}_i, \quad (23)$$

$$T^2 = \mathbf{x}_i^T \mathbf{P} \mathbf{\Lambda}^{-1} \mathbf{P}^T \mathbf{x}_i. \quad (24)$$

One can threshold statistics T^2 and Q to find outliers in principal and residual subspaces.

The thresholds are called control limits in SPM research. Qin et al. used control limits δ^2 for Q and τ^2 for T^2 from research of Box [39, 52]. Nomikos and MacGregor [53] defined δ^2 under the assumption that chi-square distribution $\chi^2(k)$ approximates Q well, which was shown by Box [52]. The $\chi^2(k)$ is a distribution of k variables that are squared and summed. The k variables follow the standard normal distribution. Qin et al. also used $\chi^2(k)$ to approximate T^2 for defining τ^2 [39]. Data sample \mathbf{x}_i is considered faulty if $Q > \delta^2$ or $T^2 > \tau^2$. Let θ_i denote the sum of the $d - l$ last eigenvalues λ_j^i . This is defined as $\theta_i = \sum_{j=l+1}^d \lambda_j^i$. Notice that the eigenvalues λ_j^i are raised to the i :th power. The control limits δ^2 and τ^2 are defined as

$$\delta^2 = \frac{\theta_2}{\theta_1} \chi^2\left(\frac{\theta_1^2}{\theta_2}\right), \quad (25)$$

$$\tau^2 = \chi^2(l). \quad (26)$$

Qin et al. stated that monitoring two statistics is harder than monitoring one statistic [39]. To alleviate the problem, Qin et al. defined a statistic φ that combines the information available in statistics Q and T^2 . The combined index φ has a control limit ς that is based on results of Box [52]. The control limit ς thresholds the combined index φ to decide if a fault has occurred. The data sample \mathbf{x}_i is flagged faulty if $\varphi > \varsigma$. The control limit is based on χ^2 distribution like δ^2 and τ^2 . The combined index φ and the control limit ς are defined as

$$\varphi = \frac{Q}{\delta^2} + \frac{T^2}{\tau^2}, \quad (27)$$

$$\varsigma = \left(\frac{\frac{l}{\tau^4} + \frac{\theta_2}{\delta^4}}{\frac{l}{\tau^2} + \frac{\theta_1}{\delta^2}} \right) \chi^2 \left(\frac{\left(\frac{l}{\tau^2} + \frac{\theta_1}{\delta^2} \right)^2}{\frac{l}{\tau^4} + \frac{\theta_2}{\delta^4}} \right), \quad (28)$$

where statistics Q , T^2 , δ^2 and τ^2 are used and the l is the number of used PCs [40].

3.6 Simulated annealing

Optimization attempts to maximize or minimize the value of function f , which is called an objective function. The objective function f measures the goodness of a solution x . In minimization, the optimal solution x_i has the lowest f value of all possible solutions $f(x_i) \leq f(x) \forall x$ [54, pp. 288]. Similarly in maximization, the optimal solution x_i has the highest f value of all possible solutions $f(x_i) \geq f(x) \forall x$. The objective function is often called a loss function or a cost function in minimization [55]. The optimal solution is called a global minimum in minimization and global maximum in maximization. They are also called as the extrema of function f . Optimization is difficult because the objective function can take a non-optimal value of local extrema. Local minimum and maximum are extrema within a threshold distance of the solution x , which is called the local neighborhood. Figure 7 plots a function $f(x) = x^3 - 3x^2 - 5x$ within a interval $[-3, 5]$. Global and local extrema are marked with red dots. The illustrated local extrema are not global extrema, but they are the optimal solutions in their local neighborhood.

Simulated annealing (SA) is a metaheuristic [54, pp. 288] for a variety of different optimization tasks. A metaheuristic is an abstract guideline of how to implement an optimization algorithm. A metaheuristic does not define the optimization task, which means that a metaheuristic can be applied in many optimization tasks. This thesis uses SA for framework optimization because SA has been successfully used to optimize classifier parameters [56–58]. SA tries to avoid local maxima and minima by intentionally and occasionally allowing sub-optimal optimization steps in minimization. A bad optimization step in minimization allows a solution that has a higher objective function value than the previous solution. The bad optimization step in minimization is defined as $f(x_{new}) > f(x_{old})$. As the random bad steps are allowed, the algorithm may be able to escape local extrema by evaluating the neighboring solutions. SA uses a temperature variable t that governs the probability of accepting a bad step during the optimization process. The probability $0 \leq P(\mathbf{x}_{new}) \leq 1$ of accepting a new solution \mathbf{x}_{new} is based on the Metropolis criterion.

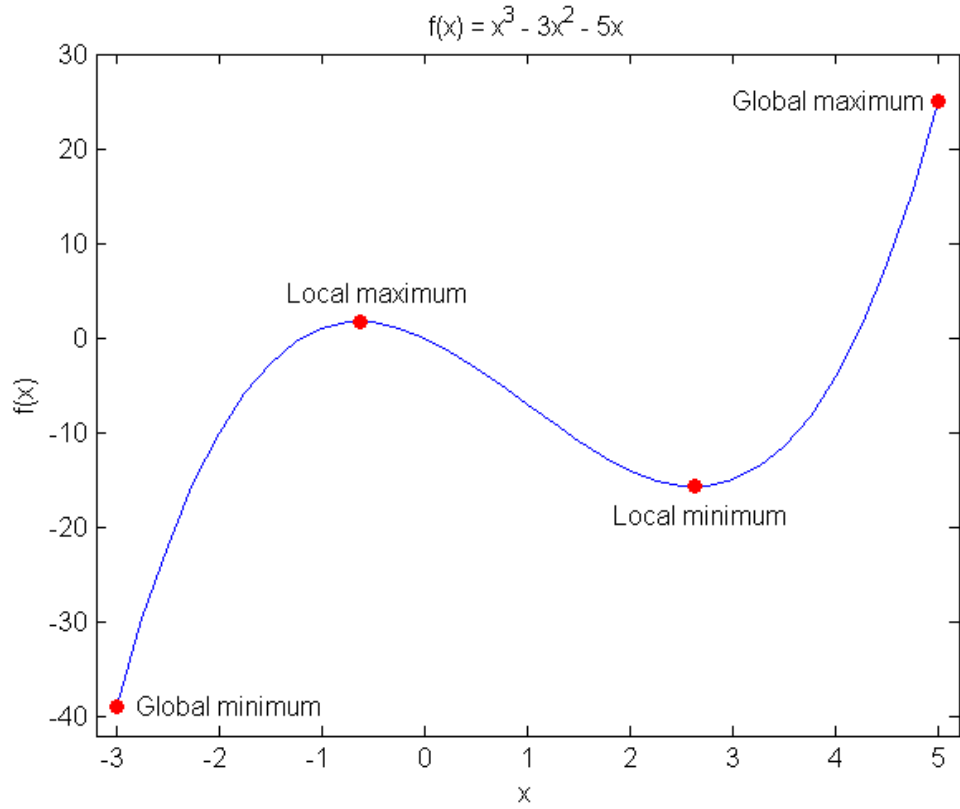


Figure 7: Plot of function $f(x) = x^3 - 3x^2 - 5x$ within interval $x = [-3, 5]$. The red dots depict function extrema. Local extrema are the extrema in their local neighborhood. Global extreme values are the extrema within the whole interval $x = [-3, 5]$.

Metropolis acceptance criterion is defined as

$$P(\text{accept } \mathbf{x}_{new}) = \begin{cases} \exp\left(\frac{-f(\mathbf{x}_{new}) - f(\mathbf{x}_{old})}{t}\right), & \text{if } (f(\mathbf{x}_{new}) - f(\mathbf{x}_{old})) > 0 \\ 1 & \text{if } (f(\mathbf{x}_{new}) - f(\mathbf{x}_{old})) \leq 0 \end{cases}, \quad (29)$$

where f is an objective function and \mathbf{x}_{new} is a new solution created from \mathbf{x}_{old} . The acceptance criterion returns 100% probability $P(\mathbf{x}_{new}) = 1.0$ if the new solution \mathbf{x}_{new} has a smaller objective function value $f(x_{new}) < f(x_{old})$. New solutions \mathbf{x}_{new} with a higher objective function value $f(x_{new}) > f(x_{old})$ acquire a probability of acceptance between 0% and 100%.

The temperature t decreases after every iteration. Lower temperature t results in lower

probability of accepting a bad optimization step. The algorithm explores a wider range of solutions in the beginning and eventually cools down in smaller range. Algorithm 2 describes the pseudocode for SA. The algorithm searches neighboring solutions for better solutions until a stopping criterion is met. [59]

Algorithm 2 Simulated annealing algorithm pseudocode.

```

1: Require: Initial solution  $\mathbf{x}$  and initial temperature value  $t > 0$ 
2: while Stopping criteria not met do
3:   Generate a new solution  $\mathbf{x}_{new}$ 
4:    $\delta = f(\mathbf{x}_{new}) - f(\mathbf{x})$ 
5:   if  $\delta \leq 0$  then
6:      $\mathbf{x} = \mathbf{x}_{new}$ 
7:   end if
8:   if  $\delta > 0 \wedge rand(0, 1) < \exp(\frac{-\delta}{t})$  then
9:      $\mathbf{x} = \mathbf{x}_{new}$ 
10:  end if
11:  Lower the temperature value  $t$ 
12: end while

```

3.7 Monte Carlo methods

Monte Carlo methods are a family of algorithms for evaluating a mathematical model repeatedly and randomly [60]. This results in an estimate of the expected value of an inspected phenomenon, which is defined by the model [61]. The algorithms rely on selecting value x with probability $P(x)$ that is defined by a probability distribution. Depending on the used probability distribution, the probability $P(x)$ might not be equal for every value x . The process of selecting values based on their probabilities is called random sampling. The approximation accuracy of the expected value increases as more samples are drawn from the system under testing [61]. A typical Monte Carlo method is implemented by executing the following steps [62, pp. 344] [60]:

1. Define a model for the simulation, which consists of a set of inputs and outputs. The model describes the relationships between the inputs and the outputs.
2. Define a probability distribution for every input. The probability distributions govern the possible values of the inputs.

3. Sample random values for the inputs with their probability distributions. The simulation model is evaluated with the sampled inputs, which results in a set of outputs.
4. Repeat the third step until a beforehand defined number of simulated outputs have been aggregated.
5. Analyze and visualize the resulting distributions of outputs.

Algorithm 3 explains an implementation of a typical Monte Carlo method in pseudocode.

Algorithm 3 Pseudocode of a Monte Carlo method.

- 1: **Require:** Probability distribution $P(x)$ for every input, a simulation model
 - 2: **while** Stopping criteria not met **do**
 - 3: Sample random values for the inputs using the probability distributions $P(x)$
 - 4: Evaluate the simulation model using the inputs
 - 5: Save the resulting outputs from the simulation model
 - 6: **end while**
 - 7: Form output distributions of the resulted output values
 - 8: Calculate descriptive statistics of the output distributions
 - 9: Visualize the output distributions
-

Testing every possible multidimensional data point has a high computational complexity. The approach of systematically testing all values is called brute-force method. For example, let the hypothetical data have 20 variables ($d = 20$) and 50 different values for every variable. The data has 50^{20} different value combinations. Using a computer that tests three billion data points per second, the brute-force method would take $1.0080 * 10^{17}$ years to complete. Studying the target system using brute-force method becomes infeasible. A Monte Carlo estimate is independent of the number of variables d and therefore one prefers to randomly sample the mathematical model [61]. Monte Carlo methods produce good results even when the curse of dimensionality affects the simulation and data.

Monte Carlo methods have many uses where one needs an approximated solution. Qin et al. used a Monte Carlo simulation [38, 39] to study the capability of PCA-based SPM to detect faults. In addition to algorithm tests, Monte Carlo methods are used for target tracking [63], financial simulations [64, 65] and solving integrals [66–68]. The following chapter defines the framework that is used to detect outliers.

4 Modular framework for outlier detection

This thesis defines and constructs a modular framework for outlier detection (MFFOD). MFFOD is a framework, which builds outlier detection applications. MFFOD couples modules, which provide the required functionalities to form an outlier detection application. The modular architecture allows application deployment for different data and environments by selecting an appropriate collection of modules. MFFOD modules originate from six different categories, which accomplish fundamental tasks in outlier detection. Figure 8 illustrates an overview of the module categories in MFFOD.

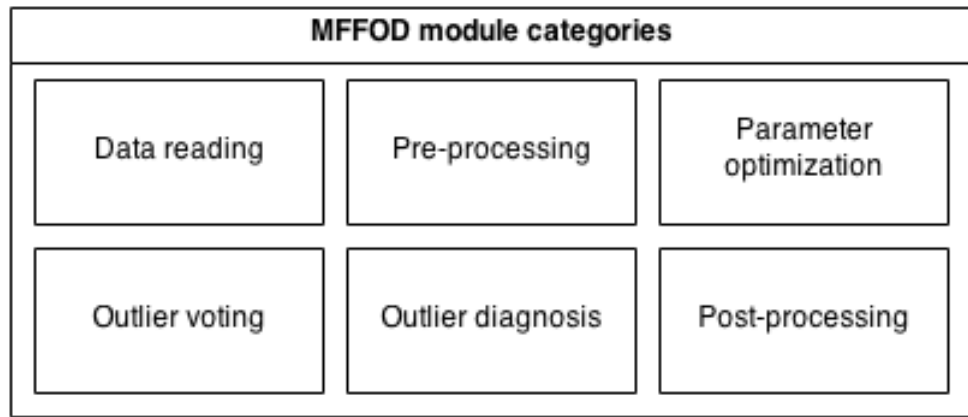


Figure 8: An overview of module categories in the modular framework for outlier detection.

The module categories are introduced in the following list.

1. Data reading

Data reading modules parse data from files, databases or data streams into usable datasets. The resulting data is contained in a data matrix X , which is also called a dataset. The data is read into computer memory for fast processing. Modules of other categories acquire data from data reading modules. Figure 9 illustrates the inputs and outputs of the data reading modules.

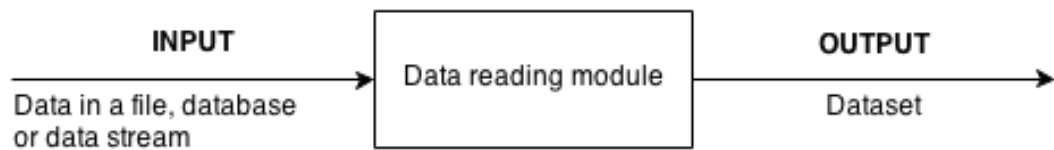


Figure 9: Input and output of a data reading module.

2. Data pre-processing

Data pre-processing modules transform data in \mathbf{X} before outlier detection is commenced. The data transform should ease or benefit the outlier detection. This is data-specific because the data governs the required pre-processing. For example, data with a high number of dimensions can use dimensionality reduction as a pre-processing module. The data is transformed within computer memory for fast processing. The resulting transformed data can be cached in a file for later reuse. Typical example of pre-processing is dimensionality reduction, which can be implemented using PCA. Figure 10 illustrates the inputs and outputs of the data pre-processing modules.

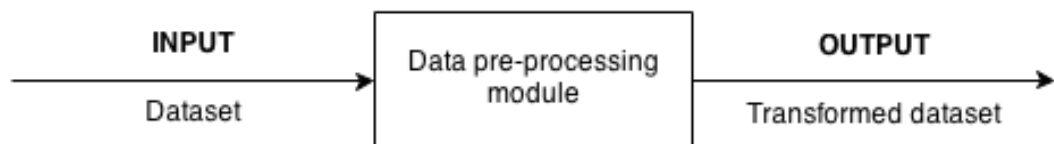


Figure 10: Input and output of a data pre-processing module.

3. Parameter optimization

Parameter optimization modules find an optimized configuration for MFFOD. The optimization improves the outlier detection of MFFOD. The optimization modules in this thesis use training data to adjust the MFFOD configuration. The training data is assumed and required to represent the characteristics of the outlier data. Chapter 4.2 describes the optimization in detail. Figure 11 illustrates the inputs and outputs

of the parameter optimization modules.

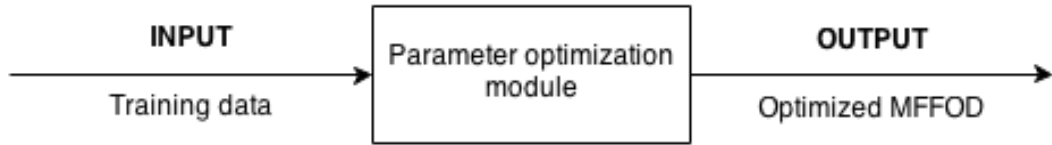


Figure 11: Input and output of a parameter optimization module.

4. Outlier voting

Outlier voting modules detect outliers in dataset \mathbf{X} and they work independently of each other. The modules inspect every data point \mathbf{x}_i in \mathbf{X} for finding outliers. The modules provide lists of indices, which point out the data points in \mathbf{X} . These indices are row numbers of \mathbf{X} . Notice that i denotes an index and a row number of a data point \mathbf{x}_i in \mathbf{X} . Outlier detection algorithms typically use algorithm-specific parameters (number of neighbors k , threshold T , ...). The voting modules implement a method for changing the values of algorithm parameters. This means that MFFOD is able to modify the parameters of outlier voting modules without having knowledge of the used algorithms. Outlier voting module is mathematically defined as a function $g(\mathbf{x}_i)$, which returns 0 if \mathbf{x}_i is an inlier and 1 if \mathbf{x}_i is an outlier. The return value of the function is formally defined as $g : \mathbb{R}^d \rightarrow \{0, 1\}$. The resulting list of outlier indices contains the index i of every data point \mathbf{x}_i with $g(\mathbf{x}_i) = 1$ in dataset \mathbf{X} . Figure 12 illustrates the inputs and outputs of the outlier voting modules.

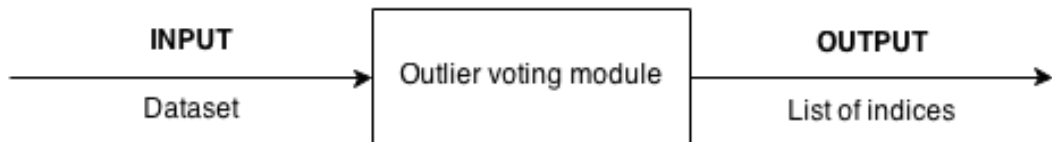


Figure 12: Input and output of an outlier voting module.

5. Outlier diagnosis

Outlier diagnosis modules investigate the principal causes of outlier detections. The diagnosis decides which variables have caused the outlier detections. Every outlier data point is diagnosed and the diagnosis information is appended to outlier detection information. Figure 13 illustrates the inputs and outputs of the outlier diagnosis modules.

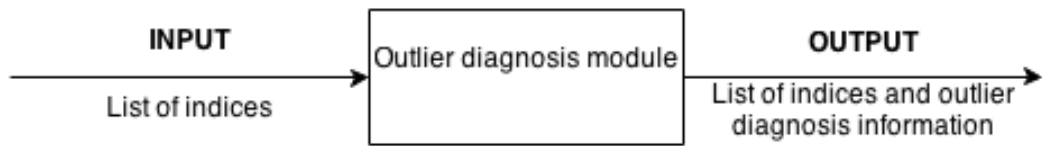


Figure 13: Input and output of an outlier diagnosis module.

6. Post-processing

Post-processing modules acquire outlier detection and diagnosis information in the final phase. The modules provide supporting functionalities, which include report generation and automated testing of the framework capability to detect outliers. Figure 14 illustrates the inputs and outputs of the post-processing modules.

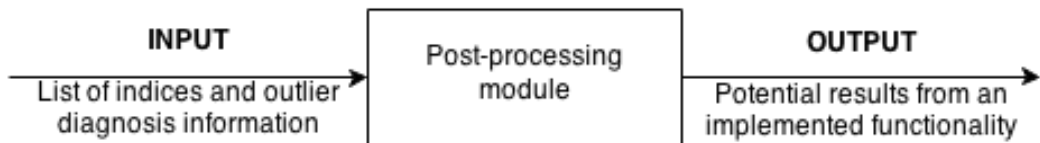


Figure 14: Input and output of a post-processing module.

The outlier detection in MFFOD is based on ensemble model with K ensemble members, which are outlier voting modules. The K outlier voting modules have corresponding weight coefficients $w_1, w_2, w_k, \dots, w_K$. The weight coefficients measure a degree of influence in outlier detection. They are interpreted as a measure of relative importance or a

measure of trust in the corresponding outlier voting modules.

The ensemble members have equal weights $w_1 = w_2 = w_k = \dots = w_K$ by default. Every ensemble member votes for outlier indices in data \mathbf{X} . The voting result for a data point \mathbf{x}_i is the sum of weight coefficients $\sum_{k=1}^K w_k * g_k(\mathbf{x}_i)$ where $g_k(\mathbf{x}_i)$ is the k :th voting module response. Voting modules affect the sum of weight coefficients if and only if $g_k(\mathbf{x}_i) = 1$. This happens when voting modules agree that \mathbf{x}_i is an outlier. Any data observation with a sum of weight coefficients $\sum_{k=1}^K w_k$ that exceeds the threshold T is detected as an outlier. This voting method is called weighted majority voting. Default value of T is K . The outlier decision function $f(\mathbf{x}_i)$ for data point \mathbf{x}_i is defined as

$$\begin{aligned} f(\mathbf{x}_i) &= \sum_{k=1}^K w_k * g_k(\mathbf{x}_i) > T, \\ f : \mathbb{R}^d &\rightarrow \{0, 1\}, \end{aligned} \tag{30}$$

where K is number of voting modules and w_k is weight of the k :th voting module. The function $f(\mathbf{x}_i)$ returns 0 if \mathbf{x}_i is an inlier and 1 if \mathbf{x}_i is an outlier. Chapter 4.2 explains the parameter optimization in detail, which automatically alters the weight coefficients.

4.1 Framework architecture

MFFOD architecture is a composition of a master controller (MC) and six layers, which correspond to the module categories. The components are

1. Master controller
2. Data reading layer
3. Data pre-processing layer
4. Parameter optimization layer
5. Outlier voting layer
6. Outlier diagnosis layer
7. Post-processing layer

MC is the main element of MFFOD, which implements the core of the framework and integrates the architectural layers. MC is the component that the programmer uses to create an application for outlier detection. MC is an interface for accessing the functionalities provided by MFFOD. MC allows the programmer to control and deploy the framework. Figure 15 illustrates the process of creating applications for outlier detection using MFFOD.

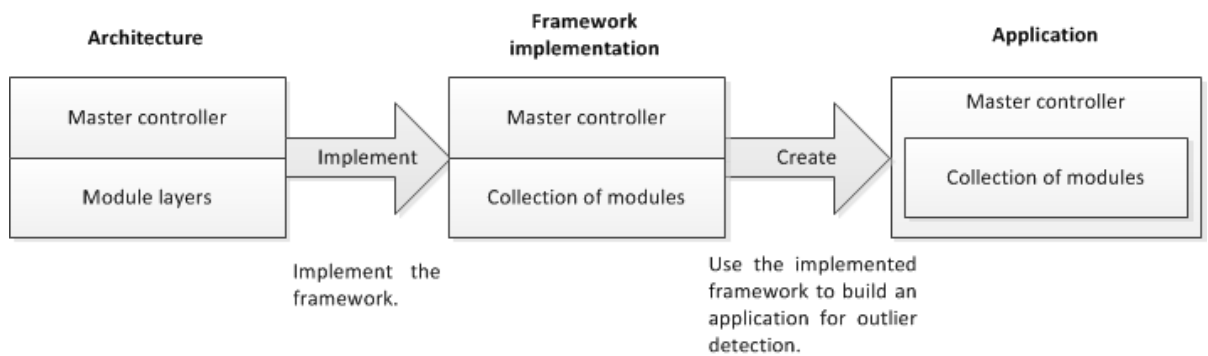


Figure 15: Framework architecture uses layers, which contain corresponding modules.

MC couples the modules and forms a working outlier detection application. The resulting application that is ready for distribution consists of a MC and a collection of modules. MC is the initiation and termination point of a MFFOD implementation because it executes every module in a MFFOD implementation. MC is responsible for executing the modules linearly in right order, which is specified in Figure 16. MC implements a method for a programmer to register modules. The modules are automatically assigned in correct layers, which are collections of modules from corresponding module categories. The automated module registration helps the programmer by abstracting the details of installing a module in MFFOD. MFFOD architecture is illustrated in Figure 16.

Every layer contributes to the outlier detection process by sharing information with the next subsequent layer. MC integrates the layers and automates the information sharing between subsequent layers. The modules within layers have specified inputs and outputs, which are defined by the module categories. The layer integration and data sharing be-

tween layers is illustrated in Figure 17. The figure is a flowchart that depicts how MC executes the layers in the correct order and handles the inputs and outputs between the modules.

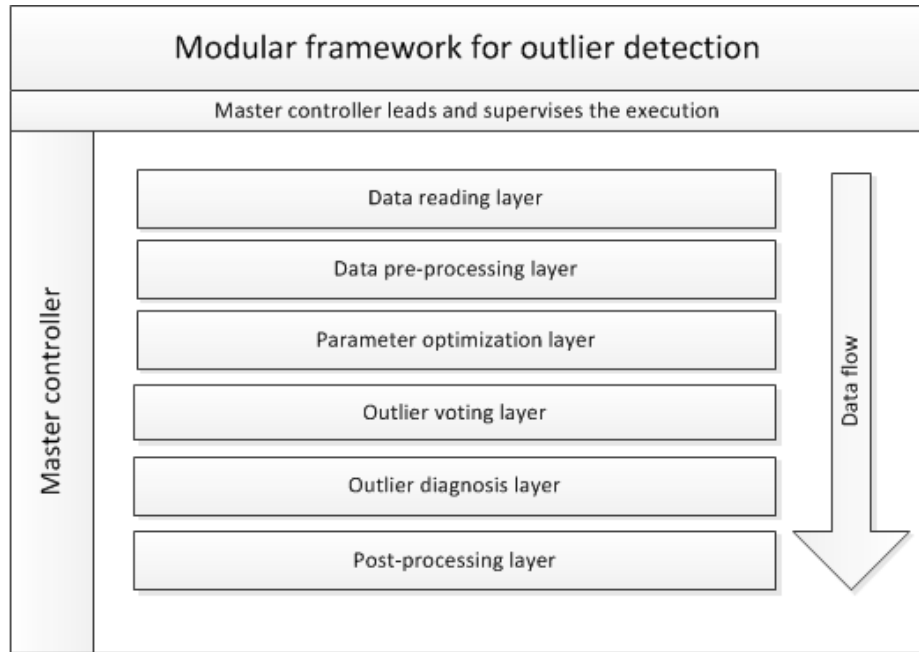


Figure 16: Framework architecture uses layers, which contain corresponding modules.

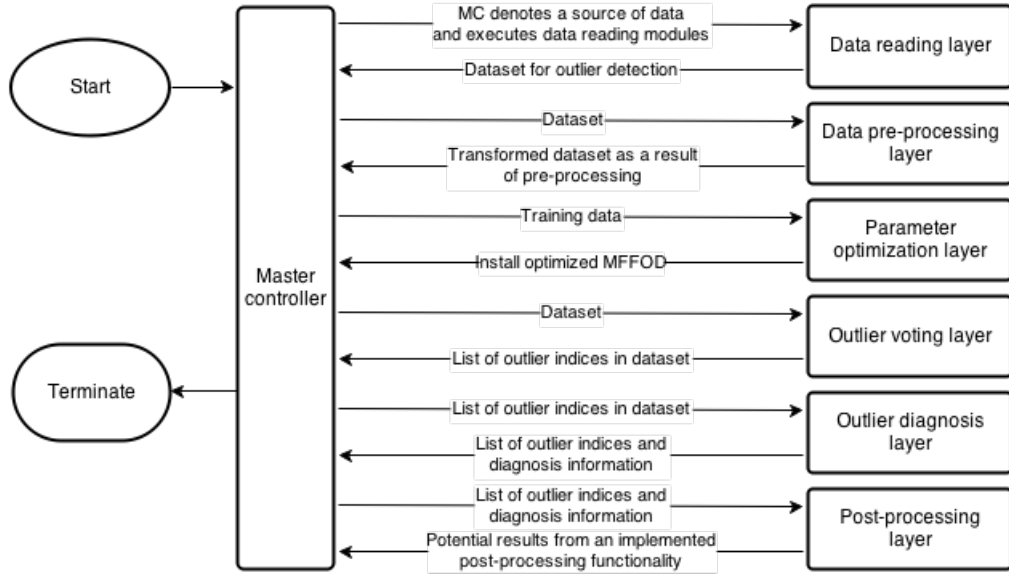


Figure 17: Flowchart of activities and information sharing between architectural components in MFFOD. MC executes the layers in linear order. MC supplies the needed data for executed modules and reads the results. Subsequent layers depend on the results of previous layers.

4.2 Parameter optimization

Whenever the MFFOD is employed in a real use case, a data analyst has to configure MFFOD. The MFFOD configuration consists of the following parameters where K is the number of ensemble members.

1. Ensemble weights $w_1, w_2, w_k, \dots, w_K$
2. Voting threshold T
3. Algorithm-specific parameters (number of neighbors k , threshold T , ...) for the K outlier voting modules

Two approaches currently exist for configuring MFFOD:

1. Manual configuration by a human expert

A human expert selects a suitable collection of outlier voting modules, which he assumes to effectively detect outliers. The human expert makes an educated guess for

selecting the initial MFFOD parameters. This requires a priori knowledge in outlier detection. The following two conditions and approaches exist for manually determining the parameters of the MFFOD.

No outlier examples are available

The initial configuration is used to detect outliers in a dataset \mathbf{X} . The human expert reviews the data points that are detected by the initial configuration. The data points that are evident outliers are saved as outlier examples. An outlier is an evident outlier if it is detected by multiple outlier voting modules. This means that the confidence of a data point \mathbf{x}_i being an outlier increases when the number of agreeing outlier voting modules ($\sum_{k=1}^K g_k(\mathbf{x}_i)$) increases.

Only few outlier examples are available

The human expert tests the MFFOD with a dataset \mathbf{X} , which has a priori known outliers. The selection of modules and parameters is refined until the MFFOD finds the known outliers. Any detected data point, which is an evident and previously unknown outlier, is added into the outlier examples.

2. Automatic configuration by an optimization method

The programmer registers an arbitrary collection of outlier voting modules in MFFOD and an optimization module. The programmer supplies the optimization module with examples of the outliers in a dataset. The optimization module automatically attempts to find MFFOD parameters, which detect the presented example outliers.

The optimization attempts to minimize the value of an objective function. The optimization knowledge is encoded within the objective function because it evaluates the goodness of a given solution. This means that objective function has to discriminate good solutions from bad solutions. Optimization solutions in optimization of MFFOD are vectors \mathbf{x} of real numbers. The solutions contain three groups of values:

1. K weights w_1, w_2, \dots, w_K
2. Voting threshold T
3. Parameters of the K outlier voting modules

The solution \mathbf{x} contains always at least $K + 1$ real numbers $(w_1, w_2, \dots, w_K, T)$. Listing 1 is an example of an optimization solution \mathbf{x} for MFFOD. The example uses three outlier voting modules. Each of the outlier voting modules uses one parameter, which is denoted by p . Symbol p_i denotes the parameter of the i :th voting module ($1 \leq i \leq K$). This results in solutions \mathbf{x} , which have a total of seven values $(w_1, w_2, w_3, T, p_1, p_2, p_3)$.

Listing 1: An example of an optimization solution \mathbf{x} for MFFOD, which combines three outlier voting modules. Each of the outlier voting modules uses one parameter. The first three values $(0.2, 0.6, 0.2)$ are the weights w_k ($w_1 = 0.2, w_2 = 0.6, w_3 = 0.2$) of the outlier voting modules. The fourth value 0.4 is the voting threshold T . The last three values $(1, 2, 3)$ are parameter values ($p_1 = 1, p_2 = 2, p_3 = 3$) of the three outlier voting modules.

```
1 0.2, 0.6, 0.2, 0.4, 1, 2, 3
```

The objective function has to be carefully crafted. The objective function in MFFOD optimization counts the number of incorrect outlier detections and missing outliers. It is important to detect as many of the outliers as possible. The number of missing outliers is doubled to emphasize the importance of detecting outliers. This means that the MFFOD prefers to detect the most of the true outliers and some of the inliers instead of being conservative and detecting some of the true outliers. An optimal configuration minimizes the objective function value because this happens when MFFOD finds every outlier without incorrect detections. The solution \mathbf{x} that produces the lowest objective function value is selected as the final optimization solution. MFFOD automatically installs the parameters that are available in the optimal solution \mathbf{x} . The following flowchart illustrates the opti-

mization process.

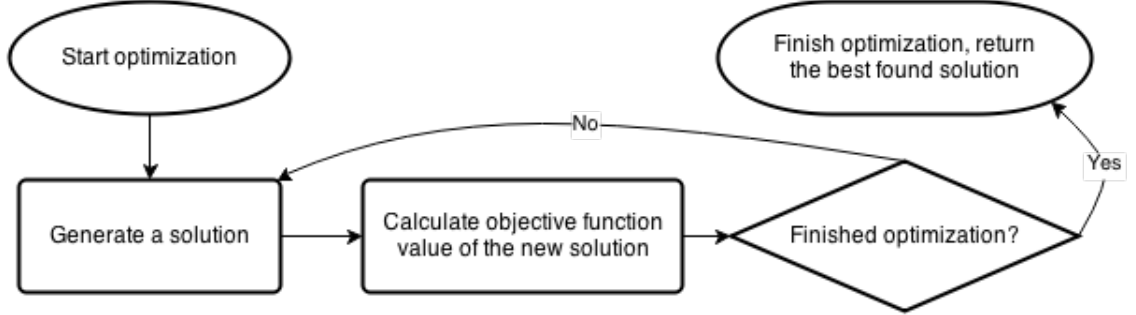


Figure 18: MFFOD optimization process. Optimization finds a solution that minimizes the objective function value. The solution that is responsible for the lowest value is selected as the final optimization solution.

For a given outlier detection problem, a set of voting modules might work better than another set of voting modules. Optimization lowers the weight coefficient w_k for less efficiently functioning voting modules. The relatively important modules typically acquire a higher coefficient value w_k . One can also consider to drop out voting modules from the evaluation, which have a relatively small w_k values. This is especially important when the performance of the outlier detection is critical. The weight coefficients can be scaled between $[0, 1]$ to ease the weight interpretation. The scaling of a weight coefficient w_k between $[0, 1]$ is defined as

$$w_k = \frac{w_k}{\sum_{i=1}^K w_i}, \quad (31)$$

where K is the number of outlier voting modules. Notice that the scaled weights sum up to one $\sum_{i=1}^K w_i = 1$.

A scaled weight w_k can be interpreted as the probability of making correct outlier detection relative to other weights. Outlier voting modules with small scaled weights can be dropped out, which can be implemented as an optimization module. This thesis does not

implement a module for dropping the outlier voting modules. No guaranteed rule exist for successful voting module selection, but analyzing the optimized weights can reveal insight about properly working outlier voting modules.

The default voting threshold T of the unoptimized MFFOD is equal to the number of voting modules K . This creates an ensemble where every voting module has to detect an outlier ($\forall k g_k(\mathbf{x}_i) = 1$) for forming the final outlier detection. This is an intersection of the lists of outlier indices from the K outlier voting modules. Even one poorly functioning voting module can undermine the outlier detection process when $T = K$ because all K votes are required. Parameter optimization configures MFFOD not to require this consensus ($\forall k g_k(\mathbf{x}_i) = 1$) between the outlier voting modules.

4.3 Data exchange between subprocesses

Outlier voting modules and diagnosis modules can be implemented as applications outside and independent of the MFFOD. Applications outside the MFFOD are called external applications. Any programming platform can be used to build the external applications. Voting and diagnosis modules can execute external applications and parse their results. The advantages of external applications include the use of code libraries from different platforms and the possibility of implementing performance-critical functionalities with highly optimized code. External applications require the data \mathbf{X} for finding outliers, which can be written in a temporary file by MFFOD. This allows external applications to acquire the analysis data. Figure 19 illustrates how MFFOD can use external applications.

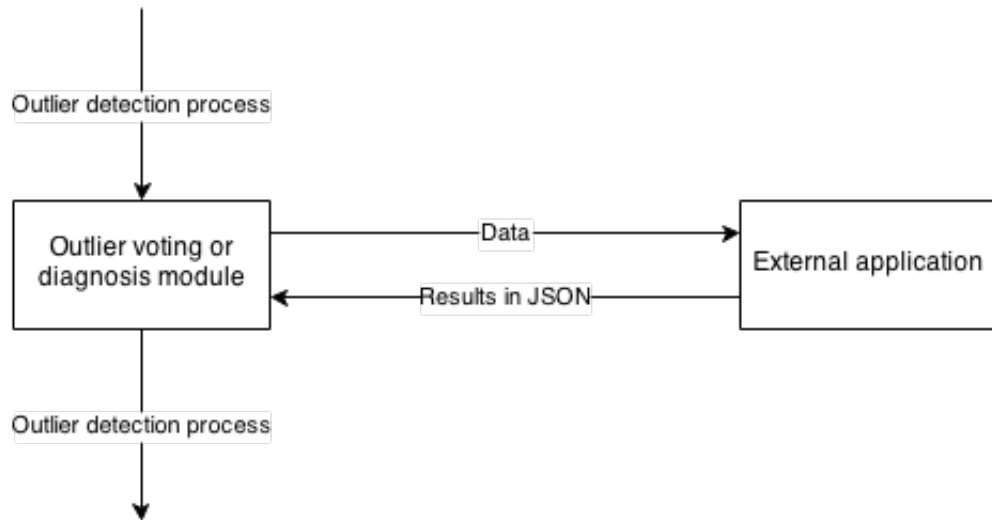


Figure 19: Outlier voting and diagnosis modules can be implemented as external applications. MFFOD modules execute external applications and read their results. The acquired results are parsed and used in the outlier detection process.

The communication between MFFOD and an external application must conform to a defined and machine-readable message format. JavaScript Object Notation (JSON) [69] was selected as the standard to transmit external application results because JSON is a textual representation of structured data. Most programming languages have libraries available for parsing JSON. External voting applications conform to the message format defined in Table 2. The format uses string “votes” as a root element and string “row” separates voted indices.

Table 2: Voting module message in JSON format.

Node	Keys	Value
Root element	votes	Array of vote elements
Vote element	row	Data index

Listing 2 is an example of a voting module message in JSON format. The example data

has three outlier indices.

Listing 2: An example of a voting module message in JSON format.

```
1 { "votes": [  
2   {"row": 121}, {"row": 27}, {"row": 52}  
3 ]}
```

Table 3 defines the message format of external diagnosis applications. The format extends the voting format in Table 2. Every vote element has an array of diagnosed variables in addition to the row numbers. The extended vote element is called diagnose element.

Table 3: Diagnosis module message in JSON format.

Node	Keys	Value
Root element	diagnosis	Array of diagnosed indices
Diagnose element	row	Data index
Diagnose element	variables	Array of diagnosed variables. The selected variables are responsible for the outlier detection.

Listing 3 is an example of a diagnosis module message in JSON format. The example data has a diagnosis of three outlier indices.

Listing 3: An example of a diagnosis module message in JSON format.

```
1 { "diagnosis" : [  
2   {"row":52, "variables": [0]},  
3   {"row":121, "variables": [2]},  
4   {"row":27, "variables": [1]}  
5 ]}
```

5 MFFOD implementation

MFFOD was implemented as a Python code library because then it can be reused and distributed conveniently. Python was selected because it has extensive numerical computing and machine learning libraries (Numerical Python, Scientific Python [50] and Scikit-learn [70]). Python is also comprehensible, fast to prototype and it is a purely object-oriented programming language. This follows the modular nature of MFFOD. The created code library provides an implementation of MC and many different modules. The programmer imports MC and any collection of modules, which are used to implement an actual MFFOD application. Every module is written in Python, except PCA-based SPM. PCA-based SPM was implemented using C++ and Eigen [49] (C++ library) as an external voting application. The external application demonstrates the MFFOD's support of using separate applications. The message format between MFFOD and the external PCA-based SPM was defined in Table 2 and in Listing 2.

Every module category implements a class in Python. The classes provide common methods within a module category. These classes are called parent classes, which allow code reuse within the categories. Actual module implementations are based on the parent classes, which is called class inheritance. The module implementations inherit a correct parent class for acquiring the category-specific common methods. The inheritance provides modularity because the module implementations can override the inherited methods. Figure 20 depicts class inheritance.

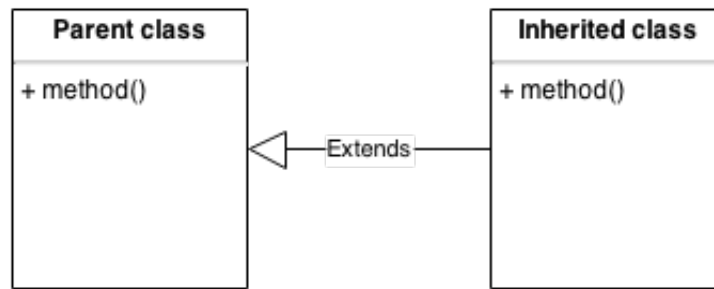


Figure 20: Example of class inheritance. The resulting child class inherits the parent class. The both classes have the same method. The inherited class can override the method implementation.

MC implementation automatically executes the common methods in parent classes. Module-specific overridden methods are also called automatically because they still have the common method names. This results in a modular plug-in architecture. MC uses a built-in Python function "type" to infer the parent class of a registered module. The module is automatically installed into the correct module layer based on the inferred parent class. This allows MC to have a single method for registering a module from any category. A unified module registration method alleviates the usage of MFFOD. The flowchart in Figure 21 illustrates how the unified module registration method works.

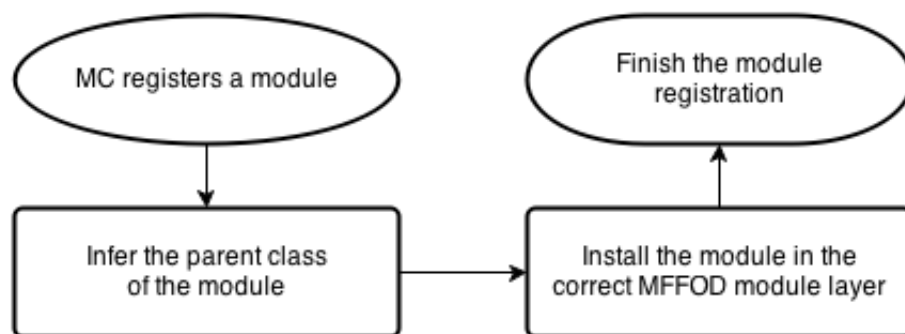


Figure 21: Flowchart of module registration in MFFOD. MC implements a unified module registration method. The method automatically installs a module from any category into a correct MFFOD layer.

5.1 Implemented modules

The MFFOD implementation uses a predefined collection of modules, which form a coherent outlier detection application. Six different modules were implemented and registered in MFFOD, which are introduced in the following listing. The listing also informs the chosen default values of parameters, which were used by the modules.

1. *Data pre-processing: PCA dimensionality reduction*

PCA dimensionality reduction projects a dataset \mathbf{X} into principal subspace. The module automatically chooses the number of PCs (l), which preserves a beforehand selected proportion of variance in the data. This explained variance of the l PCs is proportional to the sum of the corresponding eigenvalues $\sum_{i=1}^l \lambda_i$. This thesis uses a proportion of 85% explained variance because it is a frequently pre-specified proportion and a commonly known rule of thumb [71]. The selection of l for the proportion of 85% is defined as the optimization task as follows:

$$\operatorname{argmin}_l f(l) = \left| \frac{\sum_{i=1}^l \lambda_i}{\sum_{j=1}^d \lambda_j} - 0.85 \right|. \quad (32)$$

The projected data has smaller dimensionality $l < d$, which results in the dimensionality reduction.

2. *Parameter optimization: SA*

SA was selected for optimizing MFFOD because SA has been successfully used to optimize model parameters [56–58]. Default MFFOD configuration with unaltered parameters is not meant to be a production environment application, but a prototypical tool. This thesis used optimization to configure the MFFOD instead of a human expert. Testing simulations provide the outlier labels in small numbers (20 outlier examples) because real-world data does not have an abundance of pre-classified data. The parameters of the MFFOD are optimized using only a small fraction of test data, which is selected randomly. The simulations provide training with 20 outlier labels and as many inlier labels as are used in detection tests. This is one of the MFFOD

benefits because the optimization does not require a lot of a priori classified training data. The analyzed data is also selected randomly. Chapter 7 introduces and explains the experiments and used data in detail. The parameter optimization automatically selects weight coefficients w_k , voting threshold T and outlier voting module parameters.

3. *Outlier voting: LDOF*

The module implements the LDOF algorithm for detecting outliers where scores $LDOF_k(\mathbf{x}_i)$ are used to discriminate outliers from inliers. The module uses the following two parameters

- Parameter T , which is a threshold of $LDOF$ scores for classifying outliers. Default value is $T = 0.5$, which is suggested by Zhang et al. [22].
- Parameter k , which is the size of the neighborhood. Default value is $k = 10$. Breunig et al. [21] state that the k should be at least 10. Zhang et al. [22] use $k = 10$ as the smallest value of the k in their experiments with LDOF.

4. *Outlier voting: CBOD*

The module implements the CBOD algorithm for detecting outliers. The required clusters are formed using k -means clustering algorithm. The module uses the following two parameters

- Parameter ϵ , which governs the expected proportion of outliers in data. Default value is $\epsilon = 0.01$, which is suggested by Jiang et al. [31] as the upper limit of ϵ when no a priori knowledge of dataset \mathbf{X} is available.
- Parameter k , which is the number of formed clusters by k -means. Default value of k is $\sqrt{\frac{n}{2}}$ where n is the number of data points in \mathbf{X} . Mardia et al. [72] defined the rule of thumb $k = \sqrt{\frac{n}{2}}$. The resulting k is rounded to the nearest integer.

5. *Outlier voting: PCA-based SPM*

The module executes an external application, which implements PCA-based SPM for outlier detection. The external application is developed using C++ and Eigen. The

C++ compiler used "-O2"-optimization flag that allowed highly efficient vectorization of the resulting matrix calculations. The module supplies analysis data \mathbf{X} for the external application as a temporary file. The external application implements PCA using SVD. Every data point \mathbf{x}_i in \mathbf{X} is assigned a value of combined index φ , which was explained in Chapter 3.5. The SPM finds outliers in the dataset \mathbf{X}_i by using the control limit as a threshold for the combined index. The external application returns the results in the message format that was defined in Table 2 and in Listing 2. The module uses a parameter σ^2 (default value is $\sigma^2 = 0.85$), which is the percentage of explained variance by PCs. Figure 22 is a flowchart that illustrates how PCA-based SPM detects outliers.

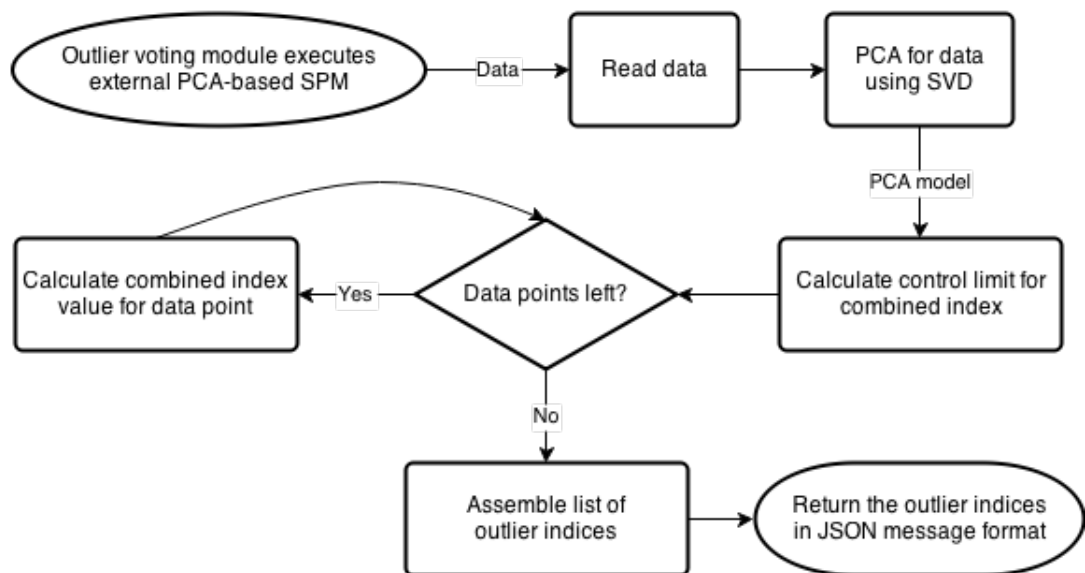


Figure 22: Flowchart that shows how PCA-based SPM detects outliers in the corresponding outlier voting module. Outliers are data points \mathbf{x}_i , which have a value of the combined index φ that exceeds the control limit ς . Both values are calculated from the data itself.

6. Post-processing: Monte Carlo testing

The module implements a Monte Carlo simulation for testing the capability of MF-FOD to detect outliers. The simulation repeatedly samples a random dataset \mathbf{X} that validates MFFOD. Every iteration produces measures of how the MFFOD performed. The measures are aggregated as distributions, which serve as statistically

significant validation results. Chapter 7 describes the experiments and the used Monte Carlo simulation in detail.

Figure 23 is a class diagram of the object-oriented approach for creating the implementation of MFFOD in Python. Every module implementation inherits a corresponding parent class. MC is a composition of the implemented modules. The class diagram illustrates how one MC can register multiple modules.

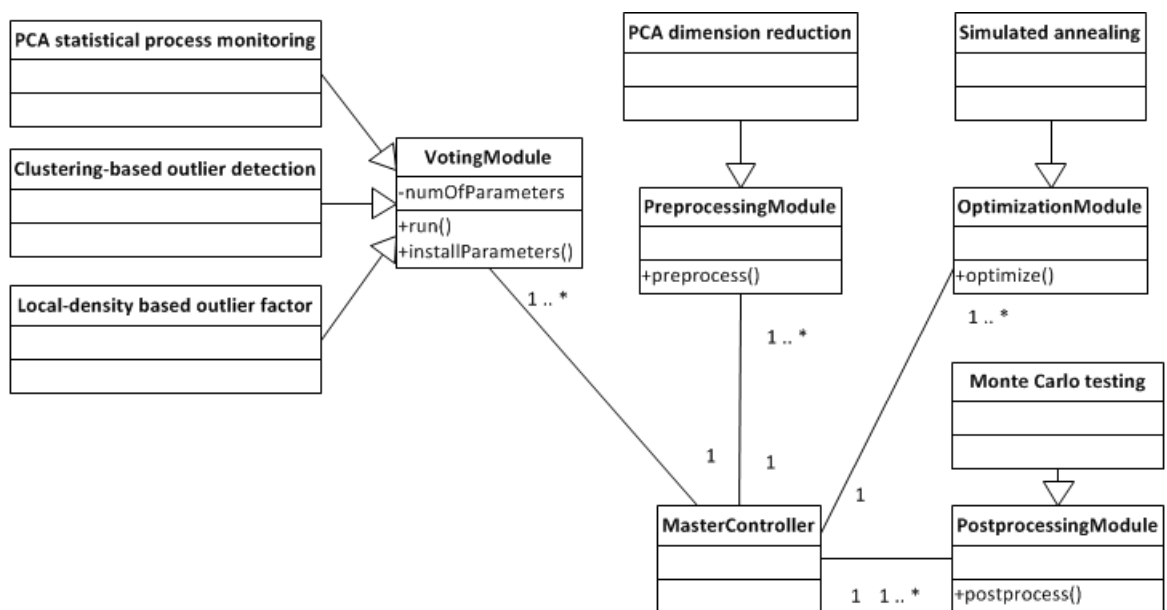


Figure 23: The MFFOD implementation class diagram. The diagram illustrates the parent classes (suffix "Module") and how the module implementations inherit the parent classes. MC is responsible for handling the modules, which is the reason why MC is a composition of multiple modules.

The MFFOD composition does not include an outlier diagnosis module because the focus of this thesis is to detect outliers. The diagnosis can be implemented by employing a measure of likeliness of variable values [73]. Variables with unlikely values are responsible for outlier detection. The likeliness can be estimated using KDE [74, 75]. Data reading modules were not used because the Monte Carlo simulation module provides the required experiment data. Chapter 7 presents the experiment data in detail.

LDOF, CBOD and PCA-based SPM were selected for outlier voting because they rely on fundamentally different approaches for detecting outliers. LDOF uses distances, CBOD uses clustering and PCA-based SPM is a quality control method. This demonstrates how MFFOD can combine arbitrary outlier detection algorithms if they adhere to the MFFOD specification. LDOF [22] and CBOD [31] are also recent algorithms in outlier research. The goal is to provide effective outlier voting modules because the goodness of the ensemble is equal to the outlier detection information diversity as different outlier detection methods have different strengths and weaknesses. MFFOD combines the strengths and mitigates the weaknesses and enables the addition of new outlier detection methods. Combining algorithms that provide the same information is not beneficial because no new information is acquired. LDOF, CBOD and PCA-based SPM were selected to provide diverse information. The next chapter discusses the properties and characteristics of the MFFOD. The benefits and novelty of the MFFOD are also discussed.

6 Characteristics of MFFOD

MFFOD provides a unified framework for developing outlier detection applications. The framework alleviates co-operation and project management within teams because the modules can be implemented as separate and independent development tasks. Ensemble learning is used to compose an outlier detection framework based on the efficient use of multiple outlier algorithms. This can reduce bias and variance of the outlier detection ensemble [12, pp. 222] [13, pp. 653]. The framework allows the reuse of the implemented modules. Researchers can build a library of modules suited for different outlier detection purposes. The framework allows rapid testing of different outlier detection algorithms with comparable results. This means that an objective head-to-head comparison of outlier voting modules is possible. The modular architecture reduces the learning curve to develop an outlier detection application because the framework has a highly abstracted programming interface. The abstraction hides the mathematical details from users. They are not expected to have expertise in outlier detection or statistics. The automatic module integration results in a working and usable outlier detection application. MFFOD can be licensed for commercial purposes without needing to publish the source code.

This thesis seems to be the first attempt to combine outlier algorithms and statistical process monitoring. It is possible as MFFOD defines only one requirement for outlier voting modules: they have to return a list of outlier indices. The actual outlier detection algorithms can be arbitrary. This means that practically every outlier algorithm can be implemented as an outlier voting module. Instead of being a purely theoretical concept, MFFOD produces fully working outlier detection applications. An application for outlier detection is a configured MFFOD with a suitable collection of modules.

Many real-world phenomena produce imbalanced data, which is data where a class dominates another class in numbers [76]. Outliers are typically vastly outnumbered by inliers because outliers are rare by definition [8]. This results in imbalanced data where inliers are heavily represented with relatively few outlier examples. Traditional supervised learning algorithms assume that a large amount of balanced and a priori known data is

available [77]. Barandela et al. [78] state that performance of supervised learning algorithms can be significantly deteriorated by using imbalanced data. Training a supervised learning algorithm with imbalanced data results in a model, which favors the dominating class [77]. This problem makes supervised learning to be an inconvenient tool for outlier detection. Semi-supervised learning methods are algorithms, which learn data with a small number of pre-classified examples and a high number of unlabeled data. Li et al. state [77] that learning imbalanced data is also challenging for semi-supervised learning algorithms because they assume to acquire balanced data. MFFOD does not attempt to learn the analysed data, unlike supervised or semi-supervised learning methods. MFFOD employs outlier detection algorithms, which are specialized in detecting outliers. MFFOD detects the outliers without modeling the intrinsic data differences between outliers and inliers. This allows MFFOD to distinguish outliers and to be optimized with imbalanced data. The following chapter describes the MFFOD implementation that was constructed in this thesis. The modules of the MFFOD implementation are based on the theoretical foundation from the third chapter. The next chapter introduces the experiments, test data, statistics and the experiment results.

7 Experiments

The goodness of MFFOD was evaluated using experiments, which measured the capability of MFFOD to detect outliers (accuracy) and the resulting execution times. The accuracy and the execution time validates the goodness of an outlier method in some outlier publications [21, 31, 79]. It is important to validate and verify that MFFOD has a satisfying accuracy because inaccurate outlier detection is not useful. It is also important to inspect the execution time because execution times could become infeasible for large datasets.

MFFOD accuracy was measured with synthetic data and public real-world datasets, which are commonly utilized in outlier publications [5, 9, 19, 22, 25, 30, 31, 33, 38, 79–81]. Synthetic datasets are artificially built datasets that do not originate from real-world environment. Real-world datasets are recorded and aggregated from real-world environments. Evaluating outlier detection methods is challenging in practice because only a few datasets exist with specifically distinguished outliers [22, 81]. This leaves three options for acquiring evaluation data: 1) generate data with outliers, 2) modify existing data or 3) sample imbalanced data from existing datasets. This thesis used the first and third option because many outlier publications sample imbalanced data [9, 18, 30, 31, 33, 79, 82] or use synthetic data [5, 9, 21, 22, 32, 33, 79, 81] to validate outlier detection. The following real-world datasets from UCI [26] machine learning repository were used because they appear often in outlier publications:

1. *Wisconsin Diagnostic Breast Cancer*

Wisconsin Diagnostic Breast Cancer (WDBC) is a medical dataset for breast cancer diagnosis. The data contains 569 data rows with 30 numerical attributes, an identification number attribute and data class. The data has two classes: benign (357 rows) and malignant (212 rows), which describe the cancer diagnosis. The numerical attributes describe the characteristics of cancer cell nuclei. The dataset is used in [9, 19, 22, 25, 30, 31, 33, 79]. Listing 4 is an example of a WDBC data row.

Listing 4: Example data row from Wisconsin Diagnostic Breast Cancer dataset. The data row describes numerical attributes of a benign breast cancer cell nuclei.

```
1 8510653, B, 13.08, 15.71, 85.63, 520, 0.1075, 0.127, 0.04568, 0.0  
    311, 0.1967, 0.06811, 0.1852, 0.7477, 1.383, 14.67, 0.004097  
    , 0.01898, 0.01698, 0.00649, 0.01678, 0.002425, 14.5, 20.49,  
    96.09, 630.5, 0.1312, 0.2776, 0.189, 0.07283, 0.3184, 0.0818  
    3
```

2. Pen-Based Recognition of Handwritten Digits

Pen-Based Recognition of Handwritten Digits (Pen-Digits) is used to classify of hand-drawn digits (0-9). The dataset contains coordinate information of 10992 drawn digits, which were acquired by drawing the digits using a pressure sensitive tablet. The digits are collected from 44 writers. The data has 16 numerical attributes and a target digit class (0-9), which describe the characteristics of the digits. The dataset is used in [9, 19, 30]. Listing 5 is an example of a Pen-Digits data row.

Listing 5: Example data row from Pen-Based Recognition of Handwritten Digits dataset. The data row describes characteristics of a hand-drawn digit four.

```
1 0, 87, 37, 100, 19, 70, 5, 42, 88, 49, 100, 58, 84, 28, 88, 0, 4
```

Synthetic data was generated to simulate sensor readings of a process with a distinctive commencement and termination. The synthetic data represents an application that is executed while six sensors provide arbitrary application-specific values. The sensors have zero values in the start and in the end because the application is not running at that time. The simulation injects 20 outliers in the data, which represent application faults. There is no established number of the injected outliers in the previous outlier research. The number of 20 outliers was chosen in this work to create imbalanced datasets because it allows a fast outlier detection. This is important because the conducted experiments are comprehensive and the computation of the experiments would take a long time with a large amount of data. The experiments with real-life datasets also used 20 outliers. A similar

approach of generating simulated process data has been used in [5, 38, 40] to validate outlier detection. Two types of outliers were used: 1) zero values and 2) spikes. Spike is an unexpectedly high sensor reading and a zero value is a sensor reading with a zero value. Figure 24 is an example of the generated synthetic process data, where the visible zero values and spikes are injected faults.

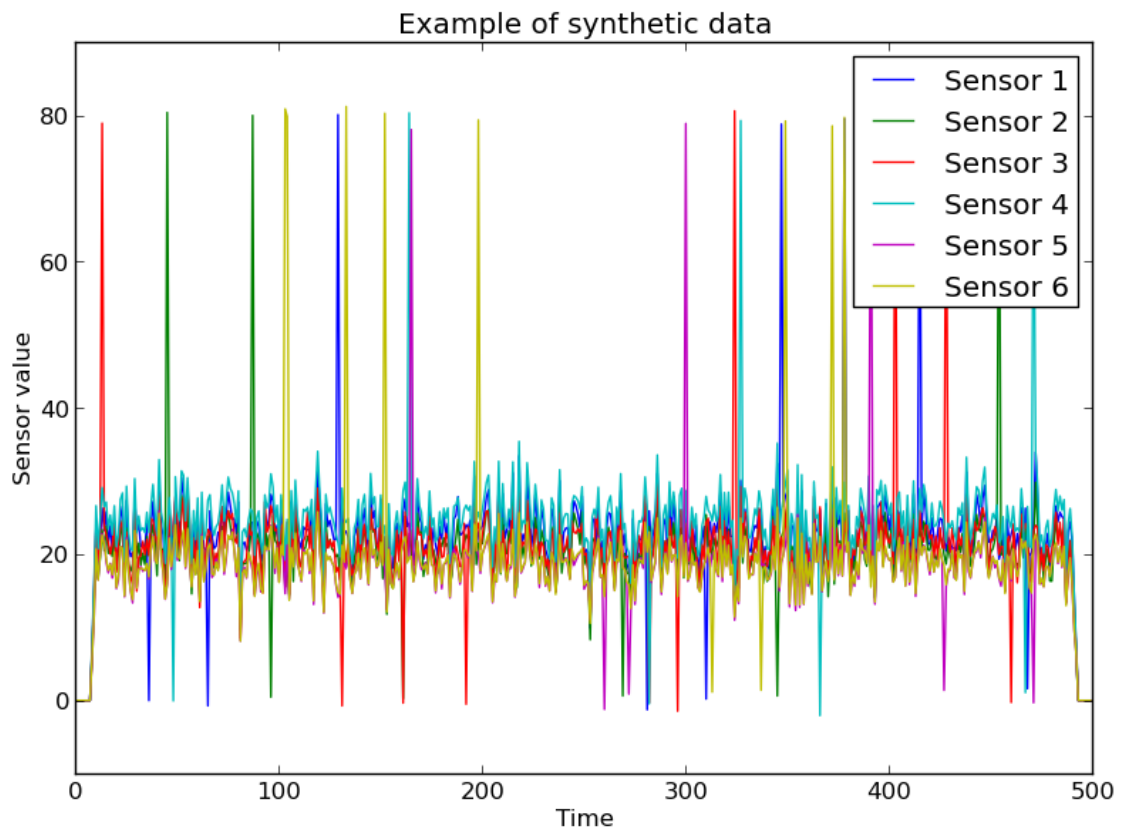


Figure 24: Example of synthetic time series data, which has a distinctive commencement and termination. This example data has 42 outliers, which are the visible sudden changes in sensor values. The outliers with zero values are contextual anomalies because a zero value is an outlier only in a specific location. The outliers with unexpectedly high values are point anomalies because a simple value thresholding can detect them. MFFOD can detect both of the outlier types by combining different outlier detection algorithms.

Many metrics for measuring performance are found in outlier research, but a typical and intuitive method is to calculate a ratio [4, 16, 19] between detected true positives and total

number of outliers. True positive is a correct outlier detection, true negative is a correct inlier detection, false positive is an incorrect outlier detection of an inlier and false negative is an incorrect inlier detection of an outlier. The four different detection types are listed in Table 4.

Table 4: Outlier detection types explained.

	Actual outlier	Actual inlier
Outlier detection	True Positive	False Positive
Inlier detection	False Negative	True Negative

Outlier detection is a binary classification task because every data observation is an outlier or an inlier. Binary classifiers assign data points in one of two available classes. The outlier status is seen as a binary value. Binary classifiers have many metrics available for measuring their performance. This thesis evaluates MFFOD by using sensitivity SEN and specificity SPC (used in [83, 84]) because they measure the capability of MFFOD to detect outliers and ignore inliers. These metrics are used to determine the capability of MFFOD to detect outliers without mistakes. Sensitivity is also called recall or true positive rate and specificity is also called false positive rate. Let TP be the number of detected true positives, TN the number of detected true negatives, FP the number of detected false positives and FN the number of detected false negatives. The metrics are defined as

$$\begin{aligned}
 SEN &= \frac{TP}{TP + FN} , \\
 SPC &= \frac{TN}{TN + FP} .
 \end{aligned}
 \tag{33}$$

Sensitivity is interpreted as the probability of finding all outliers. Specificity is interpreted as the probability of finding all inliers. Perfect outlier detection finds all outliers without false positives, which happens when $SEN = SPC = 100\%$. The metrics SEN and

SPC have to be inspected together because $SEN = 100\%$ is useless if $SPC = 0\%$ as every data index is marked as an outlier. Specificity $SPC = 100\%$ is also useless if $SEN = 0\%$ because then every data index is marked as an inlier.

The results were compared to the outlier detection capability of LOF algorithm. The LOF algorithm was used alone without optimization or preprocessing. The algorithm has two parameters: 1) the neighbourhood size k and 2) threshold T to discriminate outliers from inliers. LOF detects an outlier if LOF value exceeds the threshold T . The LOF was tested with different k and T values because the ability of LOF to detect outliers is based on appropriate parameter values. The exhaustive method of testing different parameter values is called the grid search. The grid search tests a model with different combinations of parameter values. The best parameter values are kept. The grid search finds good parameters instead of estimating or learning them. This results in more objective and fair comparison. The tested parameter values have to be limited within a range because testing all possible combinations is computationally infeasible. The MFFOD is additionally tested without optimization to test the importance of parameter optimization. An unoptimized framework is expected to have inferior outlier detection capability to an optimized framework because unoptimized MFFOD uses default parameters.

The MFFOD was validated with every combination of the three available outlier detection modules, which were

- PCA-based SPM (individually)
- LDOF (individually)
- CBOD (individually)
- PCA-based SPM and LDOF
- PCA-based SPM and CBOD
- LDOF and CBOD
- PCA-based SPM, LDOF and CBOD

A total of seven different configurations were available. Dimensionality reduction was expected to improve the outlier detection because it alleviates the curse of dimensionality. The correctness of this expectation was tested by removing PCA pre-processing module. The individual LOF that provided the baseline results was not tested with PCA pre-processing module. The additional tests without PCA pre-processing doubled the number of tests. The two public datasets and synthetic data were used in validation, which resulted in a total of 42 outlier detection tests. The 42 *SEN* and *SPC* tests were the combinations of the following parameters:

- Dataset: WDBC, pen digits, synthetic
- Outlier modules: PCA-based SPM, LDOF, CBOD
- PCA pre-processing: used, not used

Every individual test ran a Monte Carlo test with 100 iterations using imbalanced data. Testing with one iteration could have resulted in an extreme result, which would have led to false conclusions. Every iteration resulted in a pair of *SEN* and *SPC* metrics. After the simulation was terminated, the 100 pairs of *SEN* and *SPC* were aggregated as sampling distributions. The repeated tests are statistically significant and credible because the mean of the sampling distribution of the proportion is equal to the population proportion. The metrics *SEN* and *SPC* are proportions by definition. This resulted in more objective and thorough testing. The following statistics are reported in the next subsection for every test.

- Mean \overline{SEN} of the sampling distribution of *SEN*
- Mean \overline{SPC} of the sampling distribution of *SPC*
- Standard deviation $\sigma(SEN)$ of the sampling distribution of *SEN*
- Standard deviation $\sigma(SPC)$ of the sampling distribution of *SPC*

Execution time in seconds was also tested and reported for every voting module combination. The platform for the testing was a computer with 4Gb of memory and 1.73GHz

processor with eight cores. The following sections report the results of validation experiments.

7.1 Wisconsin Breast Diagnostic Cancer

WDBC was randomly sampled for creating datasets with outliers. All 30 numerical attributes were used in the tests. The benign class was regarded as inlier class and every malignant row was initially removed from the dataset. The malignant class was selected as outlier class because a benign tumor is more likely than a malignant tumor. An imbalanced subset of data was randomly sampled during every test iteration, which contained all benign records and 20 randomly chosen malignant records (5.3% of whole population). There is no established number of the randomly chosen malignant records in the earlier outlier research (10 in [9, 30], 39 in [31], 78 in [79] and 120 in [19]). MFFOD attempted to detect malignant tumors (outliers) in the imbalanced data during every test iteration.

Table 5 shows the outlier detection results with SA optimization and PCA pre-processing. A combination of LDOF and CBOD resulted in the highest $\overline{SEN} = 0.69$ where they detected on average 69% of all outliers. LDOF and CBOD also had the second highest $\overline{SPC} = 0.98$, which means that 7.14 false positives were detected on average. The highest $\overline{SPC} = 0.99$ was acquired by "PCA-based SPM and CBOD" and "PCA-based SPM, LDOF and CBOD". PCA-based SPM acquired specificity $\overline{SPC} = 0.94$ alone, which suggests that combining outlier detection algorithms reduces the number of false positive detections.

Table 5: WDBC experiment results after 100 repeats with SA optimization and PCA pre-processing. Mean and standard deviation are reported of the resulting SEN and SPC sampling distributions. The best separate results are bolded.

Configuration	\overline{SEN}	$\sigma(SEN)$	\overline{SPC}	$\sigma(SPC)$
PCA-based SPM (individually)	0.62	0.10	0.94	0.00
LDOF (individually)	0.64	0.16	0.98	0.01
CBOD (individually)	0.67	0.15	0.97	0.04
PCA-based SPM and LDOF	0.55	0.02	0.98	0.02
PCA-based SPM and CBOD	0.66	0.01	0.99	0.02
LDOF and CBOD	0.69	0.13	0.98	0.03
PCA-based SPM, LDOF and CBOD	0.61	0.01	0.99	0.02

Table 6 shows the outlier detection results with SA optimization and without PCA pre-processing. The combination of LDOF and CBOD resulted again in the highest $\overline{SEN} = 0.68$ where they detected on average 68% of all outliers. The highest $\overline{SPC} = 0.99$ was acquired by CBOD alone. All outlier voting modules and their combinations had a high specificity as the minimum \overline{SPC} was 0.98. This means that the MFFOD detected on average at most 2% of outliers as inliers. Individually used PCA-based SPM acquired the smallest sensitivity $\overline{SEN} = 0.39$ with the smallest standard deviation $\sigma(SEN) = 0.09$. This means that the individually used PCA-based SPM was the least effective without PCA pre-processing. The \overline{SEN} of the individually used PCA-based SPM was 23% smaller than its \overline{SEN} when PCA pre-processing was used. The individually used LDOF and CBOD had sensitivities $\overline{SEN} = 0.62$ and $\overline{SEN} = 0.66$. The ensemble method of combining LDOF and CBOD resulted in improved sensitivity $\overline{SEN} = 0.68$. This means that the individual LDOF and CBOD detected different outliers, which were successfully combined by the MFFOD.

Table 6: WDBC experiment results after 100 repeats with SA optimization and without PCA pre-preprocessing. Mean and standard deviation are reported of the resulting SEN and SPC sampling distributions. The best separate results are bolded.

Configuration	\overline{SEN}	$\sigma(SEN)$	\overline{SPC}	$\sigma(SPC)$
PCA-based SPM (individually)	0.39	0.09	0.98	0.01
LDOF (individually)	0.62	0.17	0.98	0.01
CBOD (individually)	0.66	0.15	0.99	0.03
PCA-based SPM and LDOF	0.56	0.19	0.98	0.02
PCA-based SPM and CBOD	0.65	0.16	0.98	0.04
LDOF and CBOD	0.68	0.16	0.98	0.03
PCA-based SPM, LDOF and CBOD	0.65	0.14	0.98	0.03

Table 7 shows the outlier detection results without SA optimization or PCA pre-processing. The outlier voting modules used their default parameters and their weight coefficients w_k had equal values. LDOF resulted in the highest sensitivity $\overline{SEN} = 1.0$. The result is ignored because the corresponding specificity \overline{SPC} was zero, which means that LDOF detected every data point. This means that the theoretical threshold $T = 0.5$ in [22] did not work well for WDBC dataset. Notice that the combinations with LDOF acquired high specificities. This means that the ensemble method was able to cope with the inefficient LDOF. PCA-based SPM resulted in the best $\overline{SEN} = 0.46$ because the corresponding specificity was $\overline{SPC} = 0.94$. This was the best result without optimization or pre-processing. The highest $\overline{SPC} = 1.0$ was acquired by every configuration except PCA-based SPM, LDOF and "PCA-based SPM and LDOF".

Table 7: WDBC experiment results after 100 repeats without SA optimization or PCA pre-preprocessing. Mean and standard deviation are reported of the resulting SEN and SPC sampling distributions. The best separate results are bolded.

Configuration	\overline{SEN}	$\sigma(SEN)$	\overline{SPC}	$\sigma(SPC)$
PCA-based SPM (individually)	0.46	0.09	0.94	0.01
LDOF (individually)	1.0	0.0	0.0	0.0
CBOD (individually)	0.11	0.04	1.0	0.0
PCA-based SPM and LDOF	0.45	0.09	0.95	0.01
PCA-based SPM and CBOD	0.09	0.04	1.0	0.0
LDOF and CBOD	0.11	0.04	1.0	0.0
PCA-based SPM, LDOF and CBOD	0.10	0.03	1.0	0.0

LOF was tested to detect outliers in imbalanced WDBC data with 100 repeats. The resulted \overline{SEN} and \overline{SPC} of LOF are tabulated in Appendices A and B. The best LOF WDBC result $\overline{SEN} = 0.49$ and $\overline{SPC} = 0.86$ was acquired in 8-neighborhood ($K = 8$) using a vote threshold $T = 1.25$. This means that the best LOF result detected 49% of outliers and 49.98 false positives ($\overline{SPC} = 0.86$) on average. The amount of false positives was high compared to MFFOD because the lowest MFFOD specificity was 0.94. LOF results show that \overline{SEN} was 100% and \overline{SPC} was 0% when voting threshold T was less than one. This was a bad result because $\overline{SPC} = 0.0$ means that every record was marked as an outlier. The \overline{SEN} acquired higher values when the size of k -neighborhood was increased. LOF had the highest \overline{SEN} with threshold T between 1 and 2. The LOF ineffectively discriminated outliers from inliers within a small range $[1, 2]$ of LOF scores. The \overline{SEN} or \overline{SPC} acquired a small score when T was outside the range $1 \leq T \leq 2$. This results in weak outlier detection capability because the LOF scores of inliers and outliers are mixed within a small range $1 \leq T \leq 2$. LOF was able to achieve relatively high \overline{SPC} when $1.5 \leq T \leq 2$, but at that point sensitivity \overline{SEN} had low scores. This means that LOF managed to ignore inliers when it did not detect outliers.

The best unoptimized PCA-based SPM result $\overline{SEN} = 0.46$ and $\overline{SPC} = 0.94$ was better than the best LOF grid search result $\overline{SEN} = 0.49$ and $\overline{SPC} = 0.86$. The resulting speci-

ficity was considerably higher. This was interesting because the default parameters are not expected to work well. The overall best result $\overline{SEN} = 0.69$ and $\overline{SPC} = 0.98$ was acquired by LDOF and CBOD MFFOD with SA optimization and PCA pre-processing. The results suggest that MFFOD outperforms LOF with WDBC data because MFFOD maintained a high \overline{SEN} and \overline{SPC} simultaneously that were higher than of LOF.

7.2 Pen-Based Recognition of Handwritten Digits

Pen-Digits was randomly sampled for creating datasets with outliers. All 16 numerical attributes were used in tests because they all contribute to the digit classification. The inlier and outlier digit classes were selected randomly during every iteration. The resulting datasets consisted of 20 outliers and 280 inliers. Kriegel et al. used 20 outliers in [9] with Pen-Digits dataset. These are imbalanced datasets because the outlier digit represents only 6.67% of the whole population.

Table 8 shows the outlier detection results with SA optimization and PCA pre-processing. Like with the WDBC, the combination of LDOF and CBOD resulted in the highest $\overline{SEN} = 0.75$ where they detected on average 75% of all outliers. LDOF and CBOD also had the highest $\overline{SPC} = 0.98$, which means that 5.6 false positives were detected on average. CBOD was the best functioning outlier detection module because the CBOD was present in all configuration with high \overline{SEN} . LDOF alone acquired $\overline{SPC} = 0.32$, which was significantly lower than CBOD $\overline{SPC} = 0.67$. Their combination resulted in the highest $\overline{SEN} = 0.75$, which proves that combining outlier algorithms can help outlier detection. This is possible because individual LDOF and CBOD detected different outliers.

Table 8: Pen-Digits experiment results after 100 repeats with SA optimization and PCA pre-preprocessing. Mean and standard deviation are reported of the resulting SEN and SPC sampling distributions. The best separate results are bolded.

Configuration	\overline{SEN}	$\sigma(SEN)$	\overline{SPC}	$\sigma(SPC)$
PCA-based SPM (individually)	0.46	0.29	0.98	0.01
LDOF (individually)	0.32	0.32	0.96	0.05
CBOD (individually)	0.67	0.42	0.96	0.06
PCA-based SPM and LDOF	0.46	0.31	0.98	0.01
PCA-based SPM and CBOD	0.7	0.39	0.97	0.04
LDOF and CBOD	0.75	0.34	0.96	0.05
PCA-based SPM, LDOF and CBOD	0.7	0.35	0.97	0.04

Table 9 shows the outlier detection results with SA optimization and without PCA pre-processing. The combination of LDOF and CBOD resulted in the highest $\overline{SEN} = 0.81$ where they detected on average 81% of all outliers. Like with the previous test with PCA pre-processing, combining LDOF and CBOD results in better \overline{SEN} than using them separately. Individual LDOF acquired a lower sensitivity $\overline{SEN} = 0.22$ than when using PCA pre-processing ($\overline{SEN} = 0.32$). Unlike LDOF, the individual CBOD acquired a higher sensitivity $\overline{SEN} = 0.76$ without using PCA pre-processing. The sensitivity $\overline{SEN} = 0.81$ of their combination was higher than the sensitivity $\overline{SEN} = 0.75$ of their combination when using PCA pre-processing. CBOD detected a larger variety of different outliers than LDOF when PCA pre-processing was not used. This explains the high sensitivity $\overline{SEN} = 0.81$ of the combined LDOF and CBOD. MFFOD successfully combined LDOF and CBOD, which detected different outliers. PCA-based SPM resulted in the highest $\overline{SPC} = 0.99$, which meant only 2.8 false positive detections on average.

Table 9: Pen-Digits experiment results after 100 repeats with SA optimization and without PCA pre-preprocessing. Mean and standard deviation are reported of the resulting SEN and SPC sampling distributions. The best separate results are bolded.

Configuration	\overline{SEN}	$\sigma(SEN)$	\overline{SPC}	$\sigma(SPC)$
PCA-based SPM (individually)	0.5	0.33	0.99	0.01
LDOF (individually)	0.22	0.28	0.98	0.02
CBOD (individually)	0.76	0.37	0.97	0.05
PCA-based SPM and LDOF	0.48	0.32	0.99	0.01
PCA-based SPM and CBOD	0.76	0.33	0.97	0.05
LDOF and CBOD	0.81	0.28	0.97	0.04
PCA-based SPM, LDOF and CBOD	0.79	0.32	0.97	0.04

Table 10 shows the outlier detection results without SA optimization or PCA pre-processing. Unoptimized LDOF was ineffective again because the resulting specificity \overline{SPC} was zero. Therefore the LDOF is ignored. PCA-based SPM and "PCA-based SPM and LDOF" resulted in the highest $\overline{SEN} = 0.45$ where they detected on average 45% of all outliers. The highest $\overline{SPC} = 1.0$ was acquired by all configurations with CBOD. This means that the configurations with CBOD did not detect inliers as outliers. The sensitivities with CBOD were very low. This is a bad result because the configurations with CBOD detected every data point as an inlier. Configurations with PCA-based SPM worked well because they detected almost 50% of all outliers with a small number of false positives.

Table 10: Pen-Digits experiment results after 100 repeats without SA optimization or PCA pre-preprocessing. Mean and standard deviation are reported of the resulting SEN and SPC sampling distributions. The best separate results are bolded.

Configuration	\overline{SEN}	$\sigma(SEN)$	\overline{SPC}	$\sigma(SPC)$
PCA-based SPM (individually)	0.45	0.29	0.98	0.01
LDOF (individually)	1.0	0.0	0.0	0.0
CBOD (individually)	0.01	0.03	1.0	0.0
PCA-based SPM and LDOF	0.45	0.28	0.98	0.01
PCA-based SPM and CBOD	0.02	0.04	1.0	0.0
LDOF and CBOD	0.0	0.01	0.01	0.03
PCA-based SPM, LDOF and CBOD	0.0	0.02	1.0	0.0

LOF was tested to detect outliers in imbalanced Pen-Digits data with 100 repeats. The resulted \overline{SEN} and \overline{SPC} of LOF are tabulated in Appendices C and D. The best LOF Pen-Digits result $\overline{SEN} = 0.24$ and $\overline{SPC} = 0.89$ was acquired in 8-neighborhood ($K = 8$) using a vote threshold $T = 1.25$. The best LOF parameters $K = 8$ and $T = 1.25$ are the same as with WDBC. This means that the best LOF result detected 24% of outliers and 30.8 false positives ($\overline{SPC} = 0.89$) on average. The Appendices C and D show that when the vote threshold exceeds one ($T \geq 1$), LOF detects outliers effectively (high \overline{SEN}) with many false positives (low \overline{SPC}).

The majority of the LOF scores were between 1 and 1.25, which has only a difference of 0.25. The LOF scores of inliers and outliers were mixed within the small range of 0.25. This means that it was hard to separate outliers from inliers because even a small change in threshold T affected the resulting sensitivity and specificity. LOF encountered this problem also with the WDBC dataset because LOF scores did not vary enough for discriminating outliers from inliers.

The best MFFOD result $\overline{SEN} = 0.81$ and $\overline{SPC} = 0.97$ was acquired with SA optimization and without PCA pre-processing using LDOF and CBOD. The best MFFOD result has higher $\overline{SEN} = 0.81$ and $\overline{SPC} = 0.97$ than the best LOF result $\overline{SEN} = 0.24$ and

$\overline{SPC} = 0.89$ acquired. Even unoptimized PCA-based SPM acquired a considerably better \overline{SEN} than the best LOF result. The results suggest that MFFOD outperforms LOF with Pen-Digits data because MFFOD maintained a high \overline{SEN} and \overline{SPC} simultaneously that were higher than of LOF.

7.3 Synthetic data

Synthetic time series data was generated with total of 300 data points. The 20 outliers were randomly inserted in the data, which resulted in imbalanced data with 6.67% outlier population. Every test iteration generated a new time series with new random outliers. Table 11 shows the outlier detection results with SA optimization and PCA pre-processing. LDOF alone resulted in the highest $\overline{SEN} = 0.85$ where it detected on average 85% of all outliers. PCA-based SPM had the highest $\overline{SPC} = 1.00$, which means no false positives were detected. This means that the individual LDOF was more efficient than the combinations of outlier voting modules.

Table 11: Synthetic data experiment results after 100 repeats with SA optimization and PCA pre-preprocessing. Mean and standard deviation are reported of the resulting SEN and SPC sampling distributions. The best separate results are bolded.

Configuration	\overline{SEN}	$\sigma(SEN)$	\overline{SPC}	$\sigma(SPC)$
PCA-based SPM (individually)	0.64	0.18	1.0	0.0
LDOF (individually)	0.85	0.18	0.99	0.02
CBOD (individually)	0.57	0.28	0.96	0.04
PCA-based SPM and LDOF	0.82	0.16	0.99	0.01
PCA-based SPM and CBOD	0.61	0.19	0.99	0.03
LDOF and CBOD	0.82	0.21	0.99	0.02
PCA-based SPM, LDOF and CBOD	0.8	0.18	0.99	0.02

Table 12 shows the outlier detection results with SA optimization and without PCA pre-processing. LDOF alone resulted again in the highest $\overline{SEN} = 0.93$ where it detected on

average 93% of all outliers. Instead of PCA-based SPM having the only $\overline{SPC} = 1.00$, now also LDOF, "PCA-based SPM and LDOF" and "PCA-based SPM and CBOD" made zero false negative detections on average. This result shows that PCA dimensionality reduction is not always useful because the results in Table 12 are better than in Table 11 (higher \overline{SEN} and \overline{SPC}). Chapter 8 discusses the reasons why PCA pre-processing might not always be beneficial. The best $\overline{SEN} = 0.93$ was also acquired without PCA pre-processing, which is considerably higher than the best result $\overline{SEN} = 0.85$ with PCA pre-processing.

Table 12: Synthetic data experiment results after 100 repeats with SA optimization and without PCA pre-preprocessing. Mean and standard deviation are reported of the resulting SEN and SPC sampling distributions. The best separate results are bolded.

Configuration	\overline{SEN}	$\sigma(SEN)$	\overline{SPC}	$\sigma(SPC)$
PCA-based SPM (individually)	0.75	0.16	1.0	0.0
LDOF (individually)	0.93	0.09	1.0	0.0
CBOD (individually)	0.7	0.21	0.97	0.03
PCA-based SPM and LDOF	0.9	0.12	1.0	0.01
PCA-based SPM and CBOD	0.74	0.12	1.0	0.02
LDOF and CBOD	0.92	0.09	0.99	0.02
PCA-based SPM, LDOF and CBOD	0.87	0.15	0.99	0.01

Table 13 shows the outlier detection results without SA optimization or PCA pre-processing. Unoptimized LDOF was ineffective again because the resulting specificity \overline{SPC} was zero. Therefore the LDOF is ignored. The combination of LDOF and CBOD resulted in the highest $\overline{SEN} = 0.11$ where they detected on average 11% of all outliers. The acquired sensitivity $\overline{SEN} = 0.11$ is an interesting result because the individual LDOF detected every data point as an outlier. This means that the ensemble method of "LDOF and CBOD" ignored the results of the individual LDOF and relied only on the individual CBOD. Every configuration except LDOF acquired perfect $\overline{SPC} = 1.00$ on average, where they made zero false negative detections.

Table 13: Synthetic data experiment results after 100 repeats without SA optimization or PCA pre-preprocessing. Mean and standard deviation are reported of the resulting SEN and SPC sampling distributions. The best separate results are bolded.

Configuration	\overline{SEN}	$\sigma(SEN)$	\overline{SPC}	$\sigma(SPC)$
PCA-based SPM (individually)	0.05	0.18	1.0	0.0
LDOF (individually)	1.0	0.0	0.0	0.0
CBOD (individually)	0.10	0.05	1.0	0.0
PCA-based SPM and LDOF	0.04	0.17	1.0	0.0
PCA-based SPM and CBOD	0.01	0.04	1.0	0.0
LDOF and CBOD	0.11	0.04	1.0	0.0
PCA-based SPM, LDOF and CBOD	0.01	0.03	1.0	0.0

LOF was tested to detect outliers in imbalanced synthetic data with 100 repeats. The resulted \overline{SEN} and \overline{SPC} of LOF are tabulated in Appendices E and F. The best LOF synthetic data result $\overline{SEN} = 0.97$ and $\overline{SPC} = 0.99$ was acquired in 8-neighborhood ($K = 8$) using a vote threshold $T = 2.0$. The neighborhood size $K = 8$ resulted in the best LOF results for all tested data. The Appendices E and F show that LOF works considerably well when voting threshold $T \geq 1.75$.

The best MFFOD result $\overline{SEN} = 0.93$ and $\overline{SPC} = 1.00$ was acquired using LDOF with SA optimization and without PCA pre-processing. MFFOD acquired slightly higher \overline{SPC} than LOF ($\overline{SEN} = 0.97$ and $\overline{SPC} = 0.99$). LOF detected outliers in synthetic data better than MFFOD because LOF had higher \overline{SEN} . The results suggest that LOF slightly outperforms MFFOD with synthetic test data because LOF detected outliers better. LOF exceeded MFFOD in outlier detection because the LOF sacrificed 1% of the perfect specificity ($\overline{SPC} = 1.00$) for detecting more outliers. MFFOD was more careful in selecting the outliers because it acquired the perfect specificity $\overline{SPC} = 1.00$. This also meant that the MFFOD detected a smaller number of the actual outliers. The undetected outliers were not as obvious outliers (low score) as the detected outliers (high score). One can also say that MFFOD preferred high specificity over high sensitivity.

7.4 Execution time tests

The execution time in seconds was measured for every outlier voting module combination. The execution time was tested using synthetic data of different sizes n . The smallest data had $n = 100$ samples while the biggest data had $n = 100100$ samples. During every iteration, the size of n was increased by 5000 samples and a new random synthetic dataset \mathbf{X} was generated. This resulted in 21 execution time tests ($n = \{100, 5100, \dots, 95100, 100100\}$) for every outlier voting module combination. Algorithm 4 describes the method of execution time testing.

Algorithm 4 Execution time testing for outlier voting modules.

```
1: while Not all outlier voting modules are tested do
2:   Set  $n = 100$ 
3:   while  $n < 100100$  do
4:     Generate a new synthetic dataset  $\mathbf{X}$  with  $n$  rows
5:     Start measuring the calculation time of outlier detection
6:     Detect outliers in  $\mathbf{X}$ 
7:     Stop measuring the calculation time of outlier detection
8:     Increment  $n$  by 5000
9:     Return the calculation time
10:  end while
11:  Aggregate the calculation times for one outlier voting module configuration
12: end while
```

Table 14 shows the resulting average number of analyzed rows per second. PCA-based SPM was the fastest configuration with 7678 average analyzed rows per second. This result was about six times faster than when using the combination of all available outlier detection modules. LDOF was the fastest outlier voting module that was implemented using Python. It was about two times slower than PCA-based SPM. Unlike the rest of the modules, PCA-based SPM was implemented using C++ and Eigen (C++-library). The acquired high speed of PCA-based SPM is an example of external application benefits.

Table 14: The average number of analyzed rows per second for different MFFOD configurations. PCA-based SPM was the fastest method for outlier detection with 7678 average analyzed rows per second. PCA-based SPM was over two times faster than LDOF, which was the second fastest method with 3393 average analyzed rows per second.

Configuration	Average analyzed rows per second
PCA-based SPM (individually)	7678
LDOF (individually)	3393
CBOD (individually)	2193
PCA-based SPM and LDOF	2005
PCA-based SPM and CBOD	1670
LDOF and CBOD	1582
PCA-based SPM, LDOF and CBOD	1266

Figure 25 plots the results of execution times for every outlier voting module combination. Every configuration exhibited a linear correlation between execution time and data size n . PCA-based SPM had the smallest execution times with all n sizes, while the execution time of the full combination escalated quickly. Table 14 and Figure 25 reveal that PCA-based SPM was the fastest outlier detection module. The combination of all three outlier detection modules was the slowest method. The combination of PCA-based SPM and CBOD had a rise and fall in the execution time with the number of rows between 6000 and 7000. This unexplained behavior is likely to be an outlier caused by unexpected operating conditions in the operating system of the test environment.

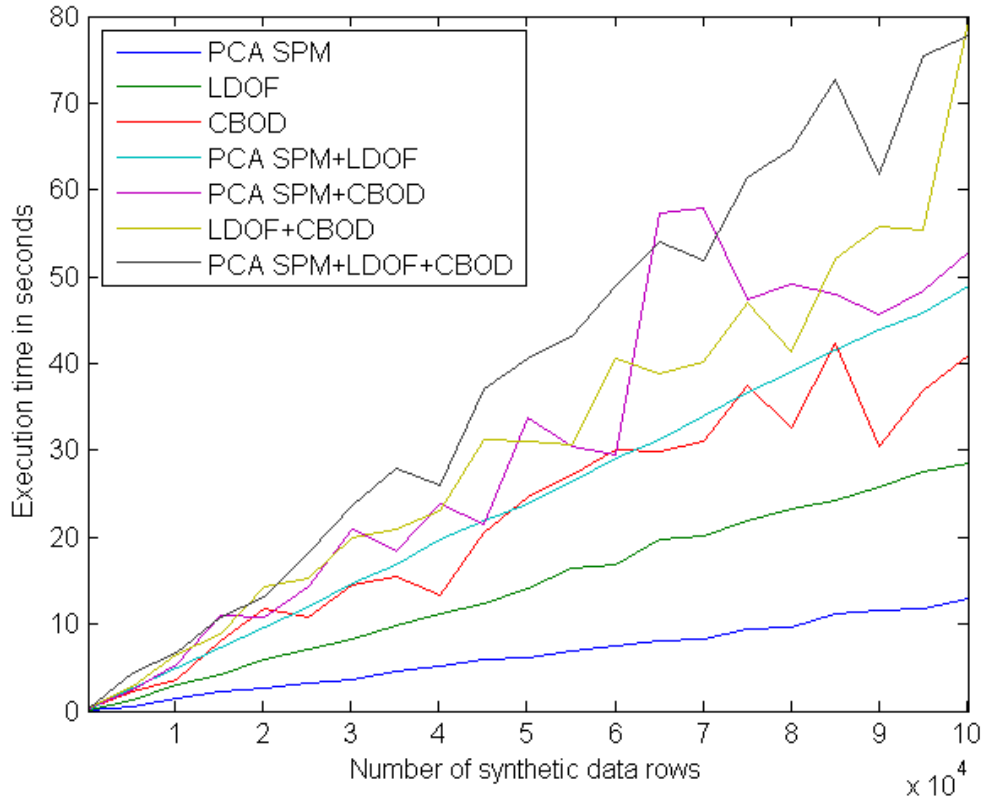


Figure 25: The plot of resulted execution times in seconds for every module configuration. The configurations exhibited a linear correlation between execution time and data size n .

8 Discussion

MFFOD was tested against the LOF algorithm using two real-world datasets (WDBC, Pen-Digits) and a synthetic dataset. The experiments showed that outlier detection algorithms can be coupled successfully. In the scope defined in this thesis, combining outlier detection algorithms improves outlier detection accuracy by increasing true positive detections and decreasing false positive detections. The conducted tests suggest that MFFOD outperforms LOF when using real-world datasets because MFFOD acquired higher sensitivity \overline{SEN} and specificity \overline{SPC} than LOF. LOF slightly outperforms MFFOD when using synthetic process data because LOF detected more outliers (higher sensitivity \overline{SEN}). MFFOD had higher specificity than LOF with synthetic data and therefore detected more inliers, but LOF detected more outliers.

The conducted experiments suggest that outlier detection modules with lower \overline{SEN} and \overline{SPC} tend to undermine the resulting \overline{SEN} and \overline{SPC} of ensemble. No outlier detection method works well with all types of data because no outlier voting module provided $\overline{SEN} \geq 0.5$ and $\overline{SPC} \geq 0.99$ for all data in experiments. No value of parameter k , T or ϵ works over all possible problems because even subsets of same data can provide different results. This is why outlier voting modules are not expected to find outliers efficiently using default parameter values. The best acquired \overline{SEN} with unoptimized MFFOD for different data were as follows:

- WDBC using PCA-based SPM $\overline{SEN} = 0.46$
- Pen-Digits using PCA-based SPM and LDOF $\overline{SEN} = 0.45$
- Synthetic data using LDOF and CBOD $\overline{SEN} = 0.11$

Optimization is beneficial in fitting outlier ensemble to find outliers in target data. Optimization significantly increased the framework performance in terms of detection accuracy. MFFOD gained significant improvement in sensitivity \overline{SEN} and specificity \overline{SPC} using optimization. The following sensitivities were acquired by an optimized MFFOD:

- WDBC using LDOF and CBOD (SA optimization, PCA dimensionality reduction) $\overline{SEN} = 0.69$
- Pen-Digits using LDOF and CBOD (SA optimization) $\overline{SEN} = 0.81$
- Synthetic data using LDOF (SA optimization) $\overline{SEN} = 0.93$

The sensitivity \overline{SEN} increased 23% for WDBC, 36% for Pen-Digits and 82% for synthetic data using optimization. This is a significant improvement in outlier detection accuracy. The results demonstrate the usefulness of optimization. PCA-based SPM had the highest sensitivities and specificities without optimization.

Dimensionality reduction using PCA did not improve the accuracy of outlier detection significantly. MFFOD configurations had higher sensitivity and specificity in Pen-Digits and synthetic data without the dimensionality reduction. The accuracy of outlier detection increased only with WDBC dataset. The combination of LDOF and CBOD acquired the WDBC sensitivity of $\overline{SEN} = 0.69$ with dimensionality reduction. The same combination acquired WDBC sensitivity $\overline{SEN} = 0.68$ without dimensionality reduction. This is only one percent boost to sensitivity, which is not a satisfactory result.

It is difficult to determine the exact reason for PCA being inefficient. One possibility is that the variance of the data in the experimented datasets is ineffectively modeled using the PCs because they are uncorrelated by definition. This requirement of having uncorrelated PCs limits the number of available loadings, which could force the PCA to explain the variance non-optimally. PCA also assumes that the d variables of a dataset \mathbf{X} are correlated, which is not always true. This is evident because the PCA uses the information available in a covariance matrix Σ . A normal distribution is defined with the covariance matrix, which is a reason why PCA works well with data that is normally distributed. The experimented datasets are not normally distributed, but follow a probability distribution that is not defined using the covariance matrix. This limits the efficiency of the PCA.

All outlier voting modules and their combinations have a linear trend in the calculation

time complexity when more analyzed data rows are added. PCA-based SPM and LDOF were the two fastest available algorithms. PCA-based SPM was clearly the fastest used outlier detection algorithm. This is a concrete example benefit of using external applications. The following section discusses about the work for future with the framework.

8.1 Future work

MFFOD opens many interesting future research questions. The following topics are ideas for future work.

How to adapt and implement outlier detection algorithms for real-time monitoring?

Instead of analyzing offline datasets, MFFOD could implement modules and framework-level support for online data monitoring. As an example, the research work could be based on SPM.

Implement MFFOD using different object-oriented languages.

The framework example was implemented using Python, but other potential object-oriented languages exist like Java. Java has a great number of code libraries, including machine learning or linear algebra required for implementing MFFOD. Broader selection of implementation environments and languages would facilitate the process of deploying MFFOD.

Invent and test novel outlier detection algorithms using the existing framework as a test environment.

Novel outlier detection algorithms can be developed inside MFFOD because a new algorithm can be implemented quickly and tested thoroughly using the existing testing methods. The resulting performance statistics can be compared to the previously acquired statistics because the testing modules implement the same tests. The outlier research community might also benefit from the unified testing capabilities and from the unified platform for conducting outlier research.

9 Conclusion

This thesis defined a set of objectives in the introduction chapter. The main objectives of this thesis were:

Specify a modular framework for outlier detection and develop an example implementation of the framework.

The modular framework for outlier detection was defined and successfully implemented. The framework managed to couple three different and independent outlier detection algorithms, which are not designed to co-operate together. Python and C++ were used in the MFFOD implementation. The MFFOD was successfully used to develop an outlier detection application.

Validate and verify the goodness of the ensemble method of three different outlier detection methods.

Three accuracy experiments and one execution time experiment were conducted to measure the MFFOD goodness. The tests were random and comprehensive with a considerable amount of repetition for statistical credibility. MFFOD acquired a good outlier detection accuracy in real-world datasets and synthetic data by combining different and independent outlier algorithms. Optimization improved the goodness of MFFOD significantly because an optimized MFFOD found more outliers than a unoptimized MFFOD. A Monte Carlo test module was implemented for the framework, which offers a unified testing environment for any MFFOD configuration.

Compile ideas for future research work.

The following section lists many ideas for future research work around MFFOD.

This thesis contributes to outlier research by defining and implementing a completely modular framework for outlier detection. The framework successfully combined arbitrary and fundamentally different outlier algorithms. Ongoing and future outlier research will automatically benefit from MFFOD and its users because novel outlier algorithms

can be developed as MFFOD modules. MFFOD is used to implement working outlier detection applications, which can be deployed into real-life use cases. This means that MFFOD is not only a theoretical concept in outlier research. The conducted tests were random, unbiased and extensive. MFFOD does not require many examples of outliers for optimization because the framework does not learn the data itself. The conducted experiments used 20 outlier examples in framework optimization.

MFFOD also leaves space for improvement. It would be beneficial to infer module parameters without labeled data. This functionality should be implemented as a separate optimization module for extended modularity. Outlier voting modules should be supplied with justified default parameter values. Effort should also be given to make the algorithms faster for handling a large amount of data. MFFOD is not unsupervised because the optimization uses examples of outliers. This is a disadvantage compared to outlier algorithms that do not use parameter values or infer them, even when MFFOD uses a small amount of examples in optimization. The resulting accuracy of outlier detection of MFFOD with small training data should be carefully studied and compared to existing supervised training algorithms.

REFERENCES

- [1] Claude Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948.
- [2] D. M. Hawkins. *Identification of Outliers*. London: Chapman and Hall, 1980.
- [3] Liu Fang and Mao Zhi-zhong. An online outlier detection method for process control time series. In *Control and Decision Conference (CCDC), 2011 Chinese*, pages 3263–3267, 2011.
- [4] P. Yogesh S. Ganapathy, N. Jaisankar and A. Kannan. An intelligent system for intrusion detection using outlier detection. *International Conference on Recent Trends in Information Technology (ICRTIT)*, pages 119–123, 2011.
- [5] W. Jun W. Tao, Z. Wenbo and Z. Hua. Workload-aware online anomaly detection in enterprise applications with local outlier factor. *Computer Software and Applications Conference (COMPSAC), 2012 IEEE 36th Annual*, pages 25–34, 2012.
- [6] E. Masciari. A framework for outlier mining in RFID data. *Database Engineering and Applications Symposium*, pages 263–267, 2007.
- [7] P. Bourgeat J. Fripp V. Villemagne C. Rowe P. Raniga, P. Schmitt and O. Salvado. Local intensity model: An outlier detection framework with applications to white matter hyperintensity segmentation. *IEEE International Symposium on Biomedical Imaging*, pages 2057–2060, 2011.
- [8] A. Banerjee V. Chandola and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 15:1–58, 2009.
- [9] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Interpreting and unifying outlier scores. In *SDM*, pages 13–24. SIAM / Omnipress, 2011.
- [10] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory, EuroCOLT '95*, pages 23–37, London, UK, 1995. Springer-Verlag.

- [11] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
- [12] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Fourth Edition*. Academic Press, 4th edition, 2008.
- [13] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
- [14] Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, KDD '05*, pages 157–166, New York, NY, USA, 2005. ACM.
- [15] Charu C. Aggarwal. Outlier ensembles: Position paper. *SIGKDD Explor. Newsl.*, 14(2):49–58, April 2013.
- [16] Bin Huang, Wen-Fang Li, De-Li Chen, and Liang Shi. An intrusion detection method based on outlier ensemble detection. In *Networks Security, Wireless Communications and Trusted Computing, 2009. NSWCTC '09. International Conference on*, volume 2, pages 600–603, April 2009.
- [17] Chunhua Ju and Na Wang. Research on credit card fraud detection model based on similar coefficient sum. In *DBTA*, pages 295–298. IEEE Computer Society, 2009.
- [18] Erich Schubert, Remigius Wojdanowski, Arthur Zimek, and Hans-Peter Kriegel. On evaluation of outlier rankings and outlier scores. In *SDM*, pages 1047–1058, 2012.
- [19] M. Bouguessa. A probabilistic combination approach to improve outlier detection. *International Conference on Tools with Artificial Intelligence (ICTAI)*, 1:666–673, 2012.
- [20] David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, New York, NY, USA, 2012.
- [21] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD*

- international conference on Management of data*, SIGMOD '00, pages 93–104, New York, NY, USA, 2000. ACM.
- [22] Ke Zhang, Marcus Hutter, and Huidong Jin. A new local distance-based outlier detection approach for scattered real-world data. *CoRR*, abs/0903.3257, 2009.
- [23] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. *SIGMOD Rec.*, 29(2):427–438, May 2000.
- [24] Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. In *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '02, pages 15–26, London, UK, 2002. Springer-Verlag.
- [25] Hoang Vu Nguyen, Hock Hee Ang, and Vivekanand Gopalkrishnan. Mining outliers with ensemble of heterogeneous detectors on random subspaces. In *Proceedings of the 15th international conference on Database Systems for Advanced Applications - Volume Part I*, DASFAA'10, pages 368–383, Berlin, Heidelberg, 2010. Springer-Verlag.
- [26] D.J. Newman A. Asuncion. UCI machine learning repository. <http://mllearn.ics.uci.edu/MLRepository.html>, 2007.
- [27] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22:85–126, 2004.
- [28] P. C. Mahalanobis. On the generalised distance in statistics. In *Proceedings National Institute of Science, India*, volume 2, pages 49–55, April 1936.
- [29] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
- [30] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Loop: Local outlier probabilities. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 1649–1652, New York, NY, USA, 2009. ACM.

- [31] Lian Duan, Lida Xu, Ying Liu, and Jun Lee. Cluster-based outlier detection. *Annals of Operations Research*, 168(1):151–168, 2009.
- [32] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B. Gibbons, and Christos Faloutsos. Loci: Fast outlier detection using the local correlation integral. In Umeshwar Dayal, Krithi Ramamritham, and T. M. Vijayaraman, editors, *ICDE*, pages 315–326. IEEE Computer Society, 2003.
- [33] E. Postma J. Janssen, F. Huszar and H. van den Herik. Stochastic outlier selection, 2012.
- [34] Huiyuan Cheng, M.P. Ooi, Ye-Chow Kuang, S. Demidenko, and B. Cheah. Outlier distribution detection approach to semiconductor wafer fabrication process monitoring. In *Quality Electronic Design (ASQED), 2011 3rd Asia Symposium on*, pages 62–67, 2011.
- [35] Yun Li, S. Nitinawarat, and V.V. Veeravalli. Universal outlier detection. In *Information Theory and Applications Workshop (ITA), 2013*, pages 1–5, 2013.
- [36] Y. Thakran and D. Toshniwal. Unsupervised outlier detection in streaming data using weighted clustering. In *Intelligent Systems Design and Applications (ISDA), 2012 12th International Conference on*, pages 947–952, 2012.
- [37] Martin Ester, Hans peter Kriegel, Jörg S, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.
- [38] Carlos F. Alcalá and S. Joe Qin. Reconstruction-based contribution for process monitoring. *Automatica*, 45(7):1593–1600, 2009.
- [39] S. Joe Qin. Statistical process monitoring: basics and beyond. *Journal of Chemometrics*, 17(8-9):480–502, 2003.
- [40] H. Henry Yue and S. Joe Qin. Reconstruction-based fault identification using a combined index. *Industrial and Engineering Chemistry Research*, 40(20):4403–4414, 2001.

- [41] Dantong Yu, Gholamhosein Sheikholeslami, and Aidong Zhang. Findout: Finding outliers in very large datasets. *Knowledge and Information Systems*, 4(4):387–412, October 2002.
- [42] Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster based local outliers. *Pattern Recognition Letters*, 2003:9–10, 2003.
- [43] Antonio Loureiro, Luis Torgo, and Carlos Soares. Outlier detection using clustering methods: a data cleaning application. In *Proceedings of the data mining for business workshop*, 2004.
- [44] Mon-Fong Jiang, Shian-Shyong Tseng, and Chih-Ming Su. Two-phase clustering process for outliers detection. *Pattern Recognition Letters*, 22(6/7):691–700, 2001.
- [45] L. J. P. van der Maaten, E. O. Postma, and H. J. van den Herik. Dimensionality Reduction: A Comparative Review. Technical report, MICC, Maastricht University, P.O. Box 616, 6200 MD Maastricht, Netherlands, February 2008.
- [46] K. Esbensen S. Wold and P. Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2:37–52, 1987.
- [47] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179—188, 1936.
- [48] V. Klema and A. Laub. The singular value decomposition: Its computation and some applications. *IEEE Transactions on automatic control*, pages 164–176, 1980.
- [49] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. <http://eigen.tuxfamily.org>, 2010.
- [50] Eric Jones, Travis Oliphant, Pearu Peterson, et al. Scipy: Open source scientific tools for python, 2001–.
- [51] MATLAB. *version 8.2 (R2013b)*. The MathWorks Inc., Natick, Massachusetts, 2013.
- [52] Box GEP. Some theorems on quadratic forms applied in the study of analysis of variance problems: Ii. the effect of inequality of variance and correlation between errors in the two-way classification, 1954.

- [53] P. Nomikos and J. F. MacGregor. Multivariate SPC charts for monitoring batch processes. *Technometrics*, 37:41–59, 1995.
- [54] Darrall Henderson, Sheldon H. Jacobson, and Alan W. Johnson. The theory and practice of simulated annealing. In Fred Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research and Management Science*, pages 287–319. Springer US, 2003.
- [55] Konrad P. Körding and Daniel M. Wolpert. Bayesian decision theory in sensorimotor control. *Trends in Cognitive Sciences*, 10(7):319 – 326, 2006. Special issue: Probabilistic models of cognition.
- [56] Randall S. Sexton, Robert E. Dorsey, and John D. Johnson. Optimization of neural networks: A comparative analysis of the genetic algorithm and simulated annealing. *European Journal of Operational Research*, 114(3):589 – 601, 1999.
- [57] Shih-Wei Lin, Zne-Jung Lee, Shih-Chieh Chen, and Tsung-Yuan Tseng. Parameter determination of support vector machine and feature selection using simulated annealing approach. *Applied Soft Computing*, 8(4):1505 – 1512, 2008. Soft Computing for Dynamic Data Mining.
- [58] Qilong Zhang, Ganlin Shan, Xiusheng Duan, and Zining Zhang. Parameters optimization of support vector machine based on simulated annealing and genetic algorithm. In *Robotics and Biomimetics (ROBIO), 2009 IEEE International Conference on*, pages 1302–1306, 2009.
- [59] F. Glover and G. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic Publishers, 1st edition, January 2003.
- [60] Samik Raychaudhuri. Introduction to monte carlo simulation. In *Proceedings of the 40th Conference on Winter Simulation, WSC '08*, pages 91–100. Winter Simulation Conference, 2008.
- [61] D. J. C. MacKay. Introduction to monte carlo methods. In *Proceedings of the NATO Advanced Study Institute on Learning in Graphical Models*, pages 175–204, Norwell, MA, USA, 1998. Kluwer Academic Publishers.

- [62] Z. Taylor and S. Ranganathan. *Designing High Availability Systems: DFSS and Classical Reliability Techniques with Practical Real Life Examples*. Wiley, 2013.
- [63] S. Yildirim, S.S. Singh, and T. Dean. A monte carlo expectation maximisation algorithm for multiple target tracking. In *Information Fusion (FUSION), 2012 15th International Conference on*, pages 2094–2101, July 2012.
- [64] Xiang Tian and K. Benkrid. Massively parallelized quasi-monte carlo financial simulation on a FPGA supercomputer. In *High-Performance Reconfigurable Computing Technology and Applications, 2008. HPRCTA 2008. Second International Workshop on*, pages 1–8, Nov 2008.
- [65] G.E. Evans and B. Jones. The application of monte carlo simulation in finance, economics and operations management. In *Computer Science and Information Engineering, 2009 WRI World Congress on*, volume 4, pages 379–383, March 2009.
- [66] Man-Zhi Li, Hong-Tao Wang, and You-Jian Shen. Improvement of calculating triple integral based on monte-carlo algorithm. In *Machine Learning and Cybernetics (ICMLC), 2010 International Conference on*, volume 4, pages 1860–1863, July 2010.
- [67] Ping Wang. Modified monte carlo method for triple integral. In *Information Technology, Computer Engineering and Management Sciences (ICM), 2011 International Conference on*, volume 1, pages 234–236, Sept 2011.
- [68] X. Legrand, A. Xemard, G. Fleury, P. Auriol, and C.A. Nucci. A quasi-monte carlo integration method applied to the computation of the pollaczek integral. *Power Delivery, IEEE Transactions on*, 23(3):1527–1534, July 2008.
- [69] Douglas Crockford. RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON). IETF RFC, <http://tools.ietf.org/html/rfc4627>.
- [70] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos,

- D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [71] H. T. Lawless and H. Heymann. *Sensory Evaluation of Food: Principles and Practices*. New York, NY, USA, 1999.
- [72] K. V. Mardia, J. T. Kent, and J. M. Bibby. *Multivariate Analysis*. Academic Press, 1979.
- [73] S. Duan and S. Babu. Automated diagnosis of system failures with Fa. In *Data Engineering, 2009. ICDE '09. IEEE 25th International Conference on*, pages 1499–1502, March 2009.
- [74] LonginJan Latecki, Aleksandar Lazarevic, and Dragoljub Pokrajac. Outlier detection with kernel density functions. In Petra Perner, editor, *Machine Learning and Data Mining in Pattern Recognition*, volume 4571 of *Lecture Notes in Computer Science*, pages 61–75. Springer Berlin Heidelberg, 2007.
- [75] Harsh K. Verma and V. S. Kumar Samparthi. Outlier detection of data in wireless sensor networks using kernel density estimation. *International Journal of Computer Applications*, 5(7):28–32, August 2010. Published By Foundation of Computer Science.
- [76] Miroslav Kubat and Stan Matwin. Addressing the curse of imbalanced training sets: One-sided selection. In *In Proceedings of the Fourteenth International Conference on Machine Learning*, pages 179–186. Morgan Kaufmann, 1997.
- [77] Shoushan Li, Zhongqing Wang, Guodong Zhou, and Sophia Yat Mei Lee. Semi-supervised learning for imbalanced sentiment classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three, IJCAI'11*, pages 1826–1831. AAAI Press, 2011.
- [78] Ricardo Barandela, José Salvador Sánchez, Vicente García, and E. Rangel. Strategies for learning in class imbalance problems. *Pattern Recognition*, 36(3):849–851, 2003.

- [79] Peng Yang and Biao Huang. Knn based outlier detection algorithm in large dataset. *Education Technology and Training and Geoscience and Remote Sensing*, 1:611–613, 2008.
- [80] Cong Li, M. Georgiopoulos, and G.C. Anagnostopoulos. Kernel principal subspace mahalanobis distances for outlier detection. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2528–2535, 2011.
- [81] Hans-Peter Kriegel, Matthias S hubert, and Arthur Zimek. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '08*, pages 444–452, New York, NY, USA, 2008. ACM.
- [82] Bo Yu, Mingqiu Song, and Leilei Wang. Local isolation coefficient-based outlier mining algorithm. In *Information Technology and Computer Science, 2009. ITCS 2009. International Conference on*, volume 2, pages 448–451, 2009.
- [83] Xuesong Lu, Yanda Li, and Xuegong Zhang. A simple strategy for detecting outlier samples in microarray data. In *Proceedings of 8th International Conference on Control, Automation, Robotics and Vision*, pages 1331–1335. IEEE, 2004.
- [84] Chetan Gupta and Robert Grossman. Outlier detection with streaming dyadic decomposition. In Petra Perner, editor, *Advances in Data Mining. Theoretical Aspects and Applications*, volume 4597 of *Lecture Notes in Computer Science*, pages 77–91. Springer Berlin Heidelberg, 2007.

Table 15: LOF WDBC data performance measured as sensitivity SEN sampling distribution (mean $\pm\sigma$) after 100 iterations. Different parameters were tested using grid search, where K is the number of effective neighbours and T is the LOF threshold for flagging an outlier. No optimization or pre-processing was used.

K/T	0.25	0.5	0.75	1	1.25	1.50	1.75	2
2	1.0 \pm 0.0	1.0 \pm 0.0	1.0 \pm 0.0	1.0 \pm 0.0	0.29 \pm 0.09	0.18 \pm 0.08	0.11 \pm 0.07	0.07 \pm 0.05
3	1.0 \pm 0.0	1.0 \pm 0.0	1.0 \pm 0.0	0.73 \pm 0.07	0.32 \pm 0.09	0.19 \pm 0.1	0.13 \pm 0.08	0.08 \pm 0.06
4	1.0 \pm 0.0	1.0 \pm 0.0	1.0 \pm 0.0	0.76 \pm 0.07	0.37 \pm 0.1	0.22 \pm 0.09	0.13 \pm 0.08	0.08 \pm 0.06
5	1.0 \pm 0.0	1.0 \pm 0.0	1.0 \pm 0.0	0.78 \pm 0.09	0.4 \pm 0.1	0.21 \pm 0.08	0.13 \pm 0.08	0.1 \pm 0.06
6	1.0 \pm 0.0	1.0 \pm 0.0	1.0 \pm 0.0	0.81 \pm 0.08	0.4 \pm 0.11	0.23 \pm 0.09	0.15 \pm 0.08	0.1 \pm 0.06
7	1.0 \pm 0.0	1.0 \pm 0.0	1.0 \pm 0.0	0.82 \pm 0.09	0.45 \pm 0.12	0.24 \pm 0.1	0.16 \pm 0.08	0.12 \pm 0.06
8	1.0 \pm 0.0	1.0 \pm 0.0	1.0 \pm 0.0	0.83 \pm 0.1	0.49 \pm 0.13	0.26 \pm 0.1	0.18 \pm 0.09	0.11 \pm 0.07
K/T	2.25	2.5	2.75	3	3.25	3.50	3.75	4
2	0.05 \pm 0.05	0.03 \pm 0.03	0.03 \pm 0.04	0.03 \pm 0.04	0.02 \pm 0.03	0.01 \pm 0.02	0.02 \pm 0.03	0.01 \pm 0.02
3	0.06 \pm 0.05	0.05 \pm 0.04	0.03 \pm 0.03	0.03 \pm 0.04	0.01 \pm 0.02	0.02 \pm 0.03	0.01 \pm 0.02	0.01 \pm 0.02
4	0.08 \pm 0.06	0.05 \pm 0.05	0.04 \pm 0.04	0.03 \pm 0.03	0.02 \pm 0.03	0.02 \pm 0.03	0.01 \pm 0.02	0.01 \pm 0.02
5	0.07 \pm 0.05	0.06 \pm 0.06	0.05 \pm 0.05	0.03 \pm 0.03	0.03 \pm 0.03	0.02 \pm 0.04	0.02 \pm 0.03	0.02 \pm 0.03
6	0.08 \pm 0.06	0.05 \pm 0.04	0.05 \pm 0.04	0.03 \pm 0.04	0.03 \pm 0.03	0.02 \pm 0.03	0.02 \pm 0.03	0.02 \pm 0.03
7	0.09 \pm 0.06	0.07 \pm 0.05	0.05 \pm 0.04	0.04 \pm 0.04	0.04 \pm 0.04	0.02 \pm 0.03	0.02 \pm 0.03	0.02 \pm 0.03
8	0.1 \pm 0.07	0.07 \pm 0.04	0.05 \pm 0.04	0.04 \pm 0.05	0.04 \pm 0.05	0.02 \pm 0.03	0.02 \pm 0.03	0.02 \pm 0.03
K/T	4.25	4.5	4.75	5	5.25	5.50	5.75	6
2	0.01 \pm 0.02	0.01 \pm 0.02	0.0 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.0 \pm 0.02	0.01 \pm 0.02	0.0 \pm 0.01
3	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.0 \pm 0.01	0.01 \pm 0.02	0.0 \pm 0.02	0.0 \pm 0.01
4	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.0 \pm 0.01	0.0 \pm 0.01	0.0 \pm 0.01	0.0 \pm 0.01
5	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.0 \pm 0.01	0.0 \pm 0.01
6	0.02 \pm 0.03	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.0 \pm 0.01	0.0 \pm 0.02	0.0 \pm 0.02
7	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02
8	0.02 \pm 0.03	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.01 \pm 0.02	0.0 \pm 0.01	0.0 \pm 0.02

Table 19: LOF synthetic data performance measured as sensitivity SEN sampling distribution (mean $\pm\sigma$) after 100 iterations. Different parameters were tested using grid search, where K is the number of effective neighbours and T is the LOF threshold for flagging an outlier. No optimization or pre-processing was used.

K/T	0.25	0.5	0.75	1	1.25	1.50	1.75	2
2	1.0 \pm 0.0	1.0 \pm 0.0	0.98 \pm 0.03	0.98 \pm 0.03	0.34 \pm 0.1	0.29 \pm 0.09	0.28 \pm 0.09	0.27 \pm 0.1
3	1.0 \pm 0.0	0.98 \pm 0.03	0.97 \pm 0.03	0.74 \pm 0.1	0.63 \pm 0.13	0.59 \pm 0.15	0.59 \pm 0.13	0.56 \pm 0.14
4	1.0 \pm 0.0	0.97 \pm 0.04	0.97 \pm 0.04	0.87 \pm 0.08	0.84 \pm 0.13	0.79 \pm 0.13	0.79 \pm 0.14	0.75 \pm 0.14
5	0.97 \pm 0.04	0.97 \pm 0.03	0.97 \pm 0.03	0.94 \pm 0.08	0.92 \pm 0.1	0.9 \pm 0.12	0.9 \pm 0.11	0.86 \pm 0.12
6	0.97 \pm 0.04	0.97 \pm 0.04	0.97 \pm 0.04	0.97 \pm 0.05	0.96 \pm 0.06	0.96 \pm 0.06	0.95 \pm 0.08	0.9 \pm 0.12
7	0.98 \pm 0.03	0.98 \pm 0.03	0.97 \pm 0.04	0.98 \pm 0.04	0.96 \pm 0.04	0.96 \pm 0.07	0.97 \pm 0.05	0.95 \pm 0.08
8	0.97 \pm 0.03	0.97 \pm 0.03	0.98 \pm 0.03	0.98 \pm 0.03	0.97 \pm 0.03	0.97 \pm 0.04	0.98 \pm 0.04	0.97 \pm 0.04
K/T	2.25	2.5	2.75	3	3.25	3.50	3.75	4
2	0.25 \pm 0.08	0.24 \pm 0.08	0.23 \pm 0.08	0.23 \pm 0.08	0.22 \pm 0.07	0.22 \pm 0.09	0.22 \pm 0.09	0.21 \pm 0.09
3	0.55 \pm 0.12	0.51 \pm 0.14	0.48 \pm 0.13	0.46 \pm 0.13	0.44 \pm 0.13	0.44 \pm 0.13	0.4 \pm 0.11	0.39 \pm 0.14
4	0.75 \pm 0.16	0.67 \pm 0.14	0.67 \pm 0.14	0.6 \pm 0.15	0.63 \pm 0.16	0.59 \pm 0.13	0.56 \pm 0.14	0.53 \pm 0.14
5	0.85 \pm 0.12	0.84 \pm 0.12	0.79 \pm 0.15	0.8 \pm 0.15	0.76 \pm 0.14	0.73 \pm 0.15	0.72 \pm 0.15	0.7 \pm 0.15
6	0.89 \pm 0.12	0.91 \pm 0.1	0.89 \pm 0.11	0.86 \pm 0.12	0.83 \pm 0.13	0.83 \pm 0.14	0.81 \pm 0.13	0.78 \pm 0.13
7	0.93 \pm 0.09	0.94 \pm 0.08	0.91 \pm 0.1	0.92 \pm 0.11	0.9 \pm 0.1	0.88 \pm 0.12	0.86 \pm 0.12	0.85 \pm 0.13
8	0.96 \pm 0.05	0.95 \pm 0.06	0.94 \pm 0.07	0.93 \pm 0.09	0.94 \pm 0.07	0.92 \pm 0.09	0.91 \pm 0.09	0.89 \pm 0.1
K/T	4.25	4.5	4.75	5	5.25	5.50	5.75	6
2	0.22 \pm 0.09	0.21 \pm 0.09	0.2 \pm 0.08	0.19 \pm 0.09	0.2 \pm 0.09	0.18 \pm 0.08	0.19 \pm 0.08	0.16 \pm 0.08
3	0.38 \pm 0.13	0.36 \pm 0.14	0.34 \pm 0.12	0.32 \pm 0.12	0.31 \pm 0.09	0.29 \pm 0.12	0.29 \pm 0.1	0.27 \pm 0.1
4	0.54 \pm 0.15	0.51 \pm 0.14	0.49 \pm 0.15	0.47 \pm 0.12	0.47 \pm 0.13	0.43 \pm 0.12	0.43 \pm 0.13	0.4 \pm 0.12
5	0.67 \pm 0.14	0.64 \pm 0.15	0.62 \pm 0.16	0.59 \pm 0.15	0.57 \pm 0.14	0.59 \pm 0.15	0.53 \pm 0.12	0.52 \pm 0.14
6	0.75 \pm 0.12	0.74 \pm 0.15	0.74 \pm 0.14	0.71 \pm 0.14	0.67 \pm 0.16	0.67 \pm 0.15	0.58 \pm 0.15	0.62 \pm 0.16
7	0.81 \pm 0.14	0.83 \pm 0.1	0.78 \pm 0.14	0.78 \pm 0.14	0.73 \pm 0.13	0.75 \pm 0.14	0.69 \pm 0.14	0.68 \pm 0.14
8	0.89 \pm 0.1	0.84 \pm 0.11	0.85 \pm 0.09	0.83 \pm 0.12	0.77 \pm 0.14	0.78 \pm 0.12	0.74 \pm 0.14	0.74 \pm 0.16

