Le, Minh-Anh:

Anomaly Detection using Machine Learning Methods Implementation and Benchmark Analysis of Selected Methods and Tuning Criteria

Master Thesis, Winter Semester 2018

Faculty of Mathematics, Computer Science and Statistics
Unspecified
Unspecified

Ludwig-Maximilians-Universität München

https://doi.org/10.5282/ubm/epub.58320

Ludwig-Maximilians-University

Master Thesis

Anomaly Detection using Machine Learning Methods

Implementation and Benchmark Analysis of Selected Methods and Tuning Criteria

Author:
Minh-Anh Le

Supervisor:
Prof. Dr. Bernd Bischl
Laura Beggel

February 2, 2018

Contents

1 Introduction **1**

2 Anomaly Detection **2**
2.1 Definition of Anomalies . . . 2
2.1.1 Nature of Input Data . . . 3
2.1.2 Types of Anomaly . . . 3
2.1.3 Availability of Labels . . . 4
2.2 Application of Anomaly Detection . . . 5
2.3 Challenges . . . 6

3 Evaluation **6**
3.1 Supervised Performance Measurement . . . 7
3.2 Unsupervised Performance Measurement: Area under the Mass-Volume Curve (AUMVC) . . . 9
3.2.1 Mass-Volume Curve (MV curve) . . . 11
3.2.2 Area under the Mass Volume Curve (AUMVC) . . . 16
3.2.2.1 Tuning Hyper parameters with AUMVC . . . 17
3.2.2.2 Computing the Volume in AUMVC . . . 17
3.2.2.3 AUMVC on High-Dimensional Data ($AUMVC_{hd}$) . 17
3.2.2.4 Choosing the Interval I for $AUMVC$ and $AUMVC_{hd}$ 19

4 Unsupervised Anomaly Detection Methods **20**
4.1 Support Vector Machine (SVM) . . . 20
4.1.1 Kernels and The Kernel-trick . . . 24
4.1.2 Support Vector Machines for Anomaly Detection . . . 25
4.1.3 Tuning . . . 27
4.2 Local Outlier Factor (LOF) . . . 28
4.2.1 Tuning . . . 31
4.3 Neural Network (NN) . . . 31
4.3.1 Neural Network for Anomaly Detection: Deep Autoencoders 35

5 Resampling Strategies for One-Class Classification **36**

6 Converting Anomaly Score Outputs into Probabilistic Outputs **37**
6.1 Probabilistic Output for Supervised Classification . . . 39
6.2 Probabilistic Output for Unsupervised Anomaly Detection . . . 40

7 Open Source `R` Package: `mlR` **43**
7.1 What is `mlR` . . . 43
7.2 General Structure of `mlR` . . . 45
7.3 Anomaly Detection in `mlR` . . . 46
7.3.1 New and modified Functions for Anomaly Detection in `mlR` . 47
7.3.2 Package `e1071` . . . 48
7.3.3 Package `kernlab` . . . 48

7.3.4 Package `DMwR` . . . 48
7.3.5 Package `dbscan` . . . 48
7.3.6 Package `h2o.deeplearning` . . . 49
7.4 Application of Anomaly Detection in `mlR` . . . 49

8 Benchmark Setup **49**
8.1 Data Sets . . . 49
8.1.1 Synthetic Data Sets . . . 50
8.1.2 Open Source Data Sets . . . 50
8.2 Method and AUMVC(hd) Settings . . . 51

9 Benchmark Results **52**
9.1 Performance and Runtime Analysis of AUMVC(hd) . . . 53
9.1.1 Runtime Analysis . . . 53
9.1.2 Fluctuation of the AUMVC(hd) . . . 58
9.1.3 AUMVC(hd) Performance w.r.t. Number of Monte-Carlo Samples . . . 60
9.1.4 AUMVC(hd) Performance w.r.t. Number of Splits of the Alpha Interval . . . 64
9.1.5 AUMVChd Performance w.r.t. to Number of Internal Subsamples . . . 66
9.2 Performance Analysis under Different Experiments Setups . . . 68
9.2.1 Tuning with Tuning Measurement AUMVC and AUMVChd . 71
9.2.1.1 AUMVC on Low Dimensional Data . . . 71
9.2.1.2 AUMVChd on High Dimensional Data . . . 73
9.2.2 Training and Tuning with Cross-Validation vs. One-Class Cross-Validation for OC-SVM . . . 75
9.2.2.1 AUMVC on Low Dimensional Data . . . 75
9.2.2.2 AUMVChd on High Dimensional Data . . . 77
9.2.3 Comparison of Methods OC-SVM and LOF . . . 78
9.2.3.1 AUMVC on Low Dimensional Data . . . 79
9.2.3.2 AUMVChd on High Dimensional Data . . . 80
9.2.4 Analysis of the Autoencoder . . . 81
9.2.4.1 AUMVC on Low Dimensional Data . . . 82
9.2.4.2 AUMVChd on High Dimensional Data . . . 85

10 Summary **87**

Appendices **94**

A Additional Measurement for Imbalanced Class Sizes **94**

B Additional Results Tables **96**

1 Introduction

Outliers or anomalies are a common occurrence in data. They can for instance result from defected measurement devices, wrongly transferred data, human error, fraud, change in population or other reasons. In the past, many methods were developed to address this issue, such as Z-score analysis, linear regression models, probabilistic and statistical modeling. But in times of increasing computational power, increasing amount of produced data, new possibilities to store data and advanced algorithms, many conventional statistical methods aren't adequate anymore. As a result, machine learning methods become increasingly important.
Detecting and differentiating anomalies from normal data is a classification problem. However, the differences of binary classification and anomaly detection are also the notable challenges, namely the lack of labels and highly unbalanced class data. In some cases, only data from one class are available. Thus, anomaly detection algorithms which can train on data containing only one class are also called one-class classification. The learning problem is then described as semi-supervised. If there is no information about the data set available at all, the learning problem is then described as unsupervised.
The first aim of this work is to analyse three different machine learning methods to detect anomalies for semi-supervised and unsupervised settings, which are the one-class support vector machine (OC-SVM), local outlier factor (LOF) and autoencoder (AE).
Moreover, performance measurements to tune hyper parameter sets and the evaluation of the used methords are of special interest. With the lack of labels, there are no possibilities to use performance measurement of classic binary classification. As it is essential to draw conclusions about the quality of used methods or it chosen hyper parameter sets an unsupervised performance measurement is investigated in this work.
The second aim of this work is to implement the anomaly detection class into the already established `mlR` package in `R` and make it available for other users.
This work starts with a general definition of the term "Anomalies" in Section 2.1 and how the application of anomaly detection methods depends on the nature of input data in Sections 2.1.1. Section 2.2 gives two short examples of real-life anomaly detection cases. Major challenges of anomaly detection are covered in Section 2.3. Since choosing an appropriate evaluation measure is one of the main challenges, Section 3.1 covers supervised measurements and Section 3.2 a new defined unsupervised measurement "Area under the Mass Volume Curve (AUMVC)". The measurement can be used on any anomaly detection methods such as the one-class SVM (Section 4.1), local outlier factor (LOF) (Section 4.2) or autoencoders (AE) (Section 4.3). To apply tuning on those methods, additional resampling strategies are introduced in Section 5. Following the algorithm in Section 6, the score value outputs of the methods are converted into probabilities to embed the methods into `mlR` and to guarantee some other advantages listed in Section 6. Section 7 gives a brief overview of the `mlR` package in `R`, what has been done to implement the anomaly detection class into mlR as well as how to use the newly implemented class. Finally, Section 8 explains the used data sets and settings for the following benchmark analysis in Section 9. The results section is divided into two parts:

the first is about investigating the AUMVC and its settings (Section 9.2), the second about using AUMVC(hd) with the previously introduced anomaly detection methods (Section 9.2.1). And lastly, Section 10 gives a summary of the results of this work.
The Appendix contains additional measurements for unbalanced class sizes (A), tables, and figures to provide more details of the results (B).

2 Anomaly Detection

Throughout the years several anomaly detection methods have been developed. The typical anomaly detection methods build models of normal data and detect deviations from the normal model in observed data [Eskin et al., 2002, p. 79]. However, to formulate the specific anomaly detection problem, the following aspects need to be taken into consideration: the nature of the input data (Section 2.1.1), type of anomalies (Section 2.1.2), availability of labels (Section 2.1.3) and constraints and requirements from the application domain [Chandola et al., 2009, p. 6]. This section starts with the definition of anomalies in Subsection 2.1, before focusing on the above mentioned aspects.

2.1 Definition of Anomalies

According to the Oxford English Dictionary, an *anomaly* is "something that deviates from what is standard, normal, or expected". In the data context, anomalies are observations, which deviate from the majority of the data [Amer et al., 2013, p. 1]. Depending on the application domain, anomalies are also referred to as outliers, discordant observations, exceptions, aberrations, surprises, peculiarities, contaminants [Chandola et al., 2009, p. 1], novelties, noise or deviations [Dau et al., 2014, p. 2].
Some popular definitions are

- "An outlying observation, or outlier, is one that appears to deviate markedly from other members of the sample in which it occurs." [Grubbs, 1969]
- "An observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data." [Hodge and Austin, 2004, p. 2]
- "An outlier is an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism." [Hawkins, 1980]
- "Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behaviour." [Chandola et al., 2009, p. 1]

There are two ways of handling anomalies, either delete or correct them, for example to yield statistically significant increase in accuracy [Smith and Martinez, 2011, p. 1] or translate them into significant actionable information [Chandola et al., 2009, p. 1].
Anomalies caused by measurement errors, for instance, are a good example of anomalies that should be deleted. In contrast to this, anomalous traffic patterns in a computer

network that could be interpreted as a hacker attack, are a good example of anomalies that should be detected to prevent it [Kumar, 2005].

In both cases, methods are needed to find these anomalies. Methods that find, delete, modify or ignore anomalies are often referred to as *noise removal* and *noise accommodation* [Hodge and Austin, 2004, p. 4]. Whereas methods that find anomalies to *extract relevant information* are referred to as *anomaly detection*. Since both categories of methods are related, the above mentioned definitions of anomaly detection are suitable for both types. The aim of anomaly detection in machine learning is to build a model that can differentiate anomalies from the remaining normal data [Dau et al., 2014, p. 2]. In this work, we will focus on anomaly detection with machine learning techniques in order to retrieve interesting information from the database.

2.1.1 Nature of Input Data

The applicability of anomaly detection methods depends on the nature of the input data, i.e., the characteristics of the input data. Input data usually contains a collection of instances which can be described as univariate, multivariate, binary, categorical or continuous [Chandola et al., 2009, p. 6]. Those attributes affect the range of methods to choose from. Moreover, the instances can have a relationship with each other, e.g., sequence data such as time series, spatial data or graph data. If no relationship among the data instances is assumed, one is dealing with record data or so-called point data. In this work we will assume to have point data with only continuous features. For these characteristics the OC-SVM (Section 4.1), LOF (Section 4.2) and AE (Section 4.3) are applicable.

2.1.2 Types of Anomaly

Before applying an anomaly detection method, it is also crucial to know which type of anomaly is present [Chandola et al., 2009, p. 8].
We can make a distinction based on whether the data contains *point anomalies*, *collective anomalies* or *contextual anomalyies*.
Point Anomaly is the simplest type of anomaly. Point anomaly is an individual data instance, which is anomalous with respect to the rest of the data [Chandola et al., 2009, p. 7]. A simple real life example is credit card fraud when only the amount spent by a person is analyzed: a transaction where the amount spent is much higher than the average amount spent is likely to be a point anomaly. But in the context of time, this kind of a transaction during Christmas instead of a summer month might be normal for that person. In the summer month, this kind of transaction would be referred to as *contextual anomaly*. Therefore, the contextual anomaly can be determined by considering behavioural attributes, e.g. amount spent within contextual attributes, such as time. [Chandola et al., 2009, p. 8]. Another type is *collective anomaly*, which only occurs in data sets with related instances. The single data points in a collective anomaly

may not be considered as anomalies by themselves, but the occurrence of these single points together indicates an anomaly [Bontemps et al., 2016, p. 2]. In the example of credit card fraud, a transaction for which the amount spent is in the normal range, is considered as normal. Now, if this amount of transaction happens every hour of a day, the occurrence together as a collection is anomalous.

2.1.3 Availability of Labels

Besides the nature of input data and anomaly type, the availability of labels is also essential when it comes to selecting the anomaly detection method. However, the availability of labels is one of the biggest challenges [Dau et al., 2014, p. 1]. Collecting labels is very expensive and requires a lot of effort, as it is often done manually by human experts [Chandola et al., 2009, p. 10]. This fact leads to three fundamental approaches to the problem of anomaly detection: *supervised*, *semi-supervised* and *unsupervised* anomaly detection.

Supervised Anomaly Detection The ideal situation would be to have a training data set with labels for anomalies and normal data. In this case, it is a classification problem [Hodge and Austin, 2004, p. 5], where the classifier learns the classification model. However, this approach is limited to what the model has seen before or what is known as anomalies, thus it requires at least enough data to cover the entire distribution of each class in order to generalize a classifier [Hodge and Austin, 2004, p. 6]. This is usually not the use case as labeled anomalous data is expensive to get [Hodge and Austin, 2004, p. 7] or the share of anomaly data is too small to derive a generalized distribution. A summary on how to handle classification and imbalanced classification problems can be found in the online tutorial of mlR (see [mlr]).

Semi-supervised Anomaly Detection In the case that the data is only labeled for the normal class, the problem can be categorized as a one-class learning problem for which semi-supervised anomaly detection techniques are needed. The main idea is to train a model for the class of normal behaviours, which is then used in the test data to identify anomalous behaviour [Chandola et al., 2009, p. 11], [Amer et al., 2013, p. 1]. This approach is analogous to the semi-supervised recognition of detection, where the model learns only the normal class and derives a boundary of normality [Hodge and Austin, 2004, p. 6]. For this approach, the methods don't require anomalous data for training but enough labeled normal data to be able to derive a generalized distribution [Hodge and Austin, 2004, p. 6]. The semi-supervised anomaly detection method is the most used one, just take for example production machines, cars or any machines, they usually run as expected and the produced data can be labeled as normal data. It is hard to simulate a situation which is not normal, and even if it would be possible to simulate this situation, there is no possibility to know if all potential situations of anomalies are

covered.
The methods one-class SVM and autoencoder are semi-supervised detection methods (see Section 4.1 and Section 4.3) and can be applied on the one-class learning problem.

Unsupervised Anomaly Detection Another common and challenging situation is to have no labeled data at all. Basically, the challenge is the non-existence of prior knowledge about the data, e.g. the question which oberservations are anomalous or what share of the data is anomalous etc. This situation is referred to as unsupervised anomaly detection [Hodge and Austin, 2004, p. 4]. To apply unsupervised anomaly detection techniques, no separation of training and testing phase is needed. Additionally [Amer et al., 2013, p. 1], one needs to make the implicit assumption that normal instances are far more frequent than anomalies [Chandola et al., 2009, p. 11] and that anomalies can be separated from the normal data [Hodge and Austin, 2004, p. 4]. The underlying concept is to model a static distribution and mark remote points as potential anomalies. For the unsupervised setting, a popular method is the local outlier factor (LOF), which is further described in Section 4.2.

2.2 Application of Anomaly Detection

Anomaly detection plays an important role in many areas such as intrusion detection, fraud detection, medical anomaly detection, industrial damage detection or anomaly detection in text [Chandola et al., 2009, p. 11].
An example of a medical anomaly detection problem is the mammography data set (Section 8.1). Mammography is one of the most effective methods for screening breast cancer. Physicians interpret the screening and decide whether a biopsy to extract sample cells or tissue is necessary to confirm the presence of breast cancer. However, 70% biopsies have a benign outcome and are therefore unnecessary [Elter et al., 2007].
In order to reduce those unnecessary biopsies, the learning algorithm should help to predict the type of tumor based on mammography screens. To train the model a data set with the attributes age, shape, margin and density can be used to predict if the mammography screen is likely to show a benign (normal) or malignant (anomaly) tumor. Most of the labeled data belongs to healthy patients, thus classification for highly unbalanced class data or semi-supervised anomaly detection methods can be used to train the model.
Another example is intrusion detection that refers to the detection of malicious activity in a computer-related system [Chandola et al., 2009, p. 12]. Especially when computer systems store sensible data (such as patient data, government data, customer data, etc.), the security for protecting the system against intrusion or so-called hacker attacks should be very high. One of the challenges of this problem is the constant modification of these attacks. Over time attackers will learn how to outsmart the detection system, and over time the detection system is improved to shield again newly evolved attacks. Since the only sure knowledge about the system is how the normal status look likes, semi-supervised or unsupervised anomaly detection methods should be preferred [Chandola et al., 2009, p. 12].

2.3 Challenges

First evidence for challenges in anomaly detection is described in the previous section when dicussing the availability of labels. The availability of labels does not only determine the method of choice, but also the limits of the the possibilities to tune the chosen model or to evaluate if the chosen model could actually detect anomalies in new, unlabeled data. Thus training, but also validation of the model is one of the major challenges in anomaly detection [Chandola et al., 2009, p. 3].
In addition, normal data as well as anomalies evolve over time in many areas, thus current anomalies but also normal data might not be representative for future observations. Furthermore, if anomalies occurred due to intervention of, e.g., hackers, these anomalies would appear to be similar to normal data. Hence it is more difficult for the model to distinguish between anomalies and normal oberservations or to define a normal training set for semi-supervised anomaly detection methods. The definition of the normal data set is another challenge, as covering all possible normal behaviour is very difficult. Additionally, anomalies in different domains might differ in their behaviour. In some domains, slight deviation from the normal behaviour is already tagged as anomalous whereas in others the same anomalies might be tagged as normal. Similar to judging if noise in the data are anomalies or normal. [Chandola et al., 2009, p. 3]
Due to those challenges, anomaly detection is a wide and underresearched area. Many anomaly detection methods and research papers are solving one specific problem for a specific characteristic of input data and domain. Those methods are developed from disciplines such as statistics, machine learning, data mining and others. [Chandola et al., 2009, p. 3]
In the following, this work will focus on semi-supervised and unsupervised machine learning methods applied to data containing point anomalies, while not taking into account the specific domain where the data comes from.

3 Evaluation

Evaluation criteria, also called performance measurements, help the user to choose an optimal method for a data set and the problem at hand. Therefore the evaluation criteria should be chosen at the beginning and used as an optimization criteron. The choice of performance measurement depends on the type of output of the anomaly detection method, which can be either scores or labels and the availability of true labels. If true labels are given the user should use supervised performance measurements (Subsection 3.1), otherwise unsupervised performance measurements (Subsection 3.2).
For methods that return scores, the analyst needs to define an optimal threshold to determine the anomalies, which can be done domain specific or model specific [Chandola et al., 2009, p.11]. For returned labels the analyst can indirectly control the threshold by controlling parameters within each method (e.g. controlling the ν variable in the OC-SVM method) [Chandola et al., 2009, p.11]. In order to find the optimal controlling parameter or threshold the user can again tune the model with an appropriate evaluation

criterion. As anomaly detection methods classify the observation in two classes (normal and anomaly), if labels are available for testing, most of the known binary classification performance measurements as well as a view anomaly detection specific performance measurements can be used (Section 3.1). There are almost no performance measurements available yet for data without labels, but one recently developed method is the Area under the Mass Volume curve (AUMVC), which is introduced in Section 3.2 and further analyzed in (Section 8).

3.1 Supervised Performance Measurement

Although unsupervised anomaly detection deals with unlabeled data, most anomaly detection papers use labels to evaluate their algorithms. In more detail, they train the model on unsupervised data but evaluate the performance of the algorithm using true labels. By doing so, a comparison over different anomaly detection algorithms is possible (e.g. Hodge and Austin [2004] or Goldstein and Uchida [2016]). But in practice, the true labels are usually not available. Section 3.2 deals with this situation.
In this section some supervised performance measurements, which are commonly used for anomaly detection, are introduced. Since we assume labeled data for evaluation, most of them are performance measurements for binary classification, that are also suitable for data with imbalanced class sizes, such as AUC, F-Score, true positive rate and true negative rate (e.g. Campos et al. [2016]). Nevertheless, not all measurements derived from the confusion matrix are adequate for the specific case of anomaly detection. For instance, the accuracy measure ACC defined as $(TP + TN)/N$ (see definition in Figure 1) is not suitable for imbalanced classes. In anomaly detection we usually expect a maximal share of 10% of anomalies, therefore the TP is comparatively small to the TN and the proportion between TP and TN is highly unbalanced. Thus, although in case the algorithm performs worse in detecting anomalies, its ACC can still yield a high value. For this reason, Campos et al. [2016, p.900] introduce additional performance measurements such as the weighted accuracy, top-p accuracy, precision at n, adjusted precision at n or average precision at n, which will be explained in more detail in Appendix A and are also implemented in `mlR`.
For the benchmark analysis, we only focus on the AUC as supervised performance measurement for anomaly detection.
Overview of some notations in this section:

- O: set of true anomalies or outliers in the data set
- I: set of true normal observations

Confusion matrix In supervised classification, a confusion matrix is often used for evaluation, as it indicates whether the algorithm confuses two classes. An overview of the content of a confusion matrix and measures that can be derived from it are displayed in Figure 1.

		True Class		
	Total Population (N)	Anomaly/Positive Class (P)	Normal/Negative Class (N)	
Predicted Class	Predicted Anomaly/Positive (PP)	True Positive (TP)	False Positive (FP)	Positive Predictive Value (PPV) = TP/PP
	Predicted Normal/Negative (PN)	False Negative (FN)	True Negative (TN)	Negative Predicted Value (NPV) = TN/PN
		True Positive Rate (TPR) = TP/P	False Positive Rate (FPR) = FP/N	F1-Score = 2 / (1 / TPR + 1/PPV)
		False Negative Rate (FNR) = FN/P	True Negatie Rate (TNR) = TN/N	G-mean = sqrt(TPR x PPV)
		Balanced Accuracy = (TPR+TNR)/2		
		Weighted Accuracy = p x TPR + (1-p) * TNR, with $0 \leq p \leq 1$		
		Matthews Correlation Coefficient (MCC) = (TP x TN - FP x FN) / sqrt[(TP + FP) (TP + FN) (TN + FP) (TN + FN)]		

Figure 1: A confusion matrix, partly taken out from Fawcett [2006, p.2]. The positive class is the anomaly class (P), the negative class the normal class (N).

Most of the measures in Figure 1 are commonly known and self-explanatory Fahrmeir et al. [2016], the rest is shortly described in the following:

- The *F1* does not take the TN (=normal) class into account, which is favorable as the focus in anomaly detection is on detecting anomalies [Garcıa et al., 2009, p.1].$F1 \in [0, 1]$, where 1 is indicating best prediction.
- The *G-mean* is the geometric mean of TPR and PPV and is also considered as a measure suitable for imbalanced data [Kubat et al., 1997, p.2]. $G\text{-}mean \in [0, 1]$, where 1 indicates best prediction.
- *Bac* is also a performance measure for skewed class distributions [Garcıa et al., 2009, p.1], that compares the number of positives and negatives relative to their class sizes (TPR and TNR). $Bac \in [0, 1]$, where 1 indicates best prediction.
- The *wac* is a weighted *bac*. For weights $w = 0.5$ the *wac* is the *bac*. Although *bac* is already suitable for imbalanced data, the user can focus more on the detection of anomalies (positive class) than the detection of the normal class (negative class).

ROC and AUC The Receiver-Operating-Characteristic-Curve (ROC) is a graphical method to evaluate the performance of an algorithm. ROC plots the TPR against the FPR [Powers, 2011, p.4]. Algorithms yielding coordinates in the far top left corner, which means having a FPR of 0 and a TPR of 1, are the best classifiers (and vice versa for the worse classifiers). Algorithms that classify as good as random guessing will score along the diagonal, where $TPR = FPR$ [Powers, 2011, p.4]. Therefore, a classifier whose ROC curve is strictly above the ROC of another classifier is considered to perform better in terms of the ROC. Thus the ROC is a tool to compare different classifiers [Powers, 2011, p.4]. In cases where two ROC curves intersect, it is sometimes hard to determine which classifier performs better. In case the user wants to perform tuning with several different settings, a visual evaluation isn't effective nor accurate enough. Instead of using the ROC curve we can summarize the ROC in one value, the *area under the curve* (AUC). The classifier that has a higher AUC value performs better. The ROC and AUC are insensitive to imbalanced data, as they consider TPR and FPR. As the AUC is one

of the most popular measures, which is widely used in other anomaly detection papers, we will use it as the benchmark measure for the experiments in Section 8.

3.2 Unsupervised Performance Measurement: Area under the Mass-Volume Curve (AUMVC)

One of the key challenges in unsupervised learning is to evaluate performance without having access to labels. Until today, there is no established way to measure performance in unsupervised learning. Most of the papers dealing with anomaly detection build the model on data pretending to don't have labels, but evaluate on data with labels, as their objective is simply to compare different algorithms in a general manner, see for example Goldstein and Uchida [2016]. Research is beginning to develop measurements to address the problem of unsupervised performance measures. For example the algorithm in Thomas et al. [2016] and Goix [2016], both based on the theory of the *Mass-Volume Curve (MV curve) for scoring functions* from which they extract a scalar measurement value called the *Area under the Mass-Volume curve*. It can be applied on low dimensional data (AUMVC) with less than 8 features or high dimensional data (AUMVChd). The AUMVC is specifically developed for tuning in the unsupervised setting.
Figure 2 is a visualization of the problem of unsupervised learning of hyper parameter, it shows the sorted scores for the data sets *5perc* and *banana* (see Section 2.1.3). In unsupervised learning, it is usually not possible to color the data points, (anomalies red and the normal observation blue). Imagine we would set a threshold based on this two graphics (without colors). For the left we would choose a threshold at -50 and if we have the labels, we would know that the chosen threshold is an appropriate one. For the right graphic we would choose a threshold around -500. With labels, one can verify that there isn't any appropriate threshold, as the anomaly detection algorithm was already failing to calculate a correct scoring function to map the data. But as labels are not available, we wouldn't know how appropriate the threshold is.

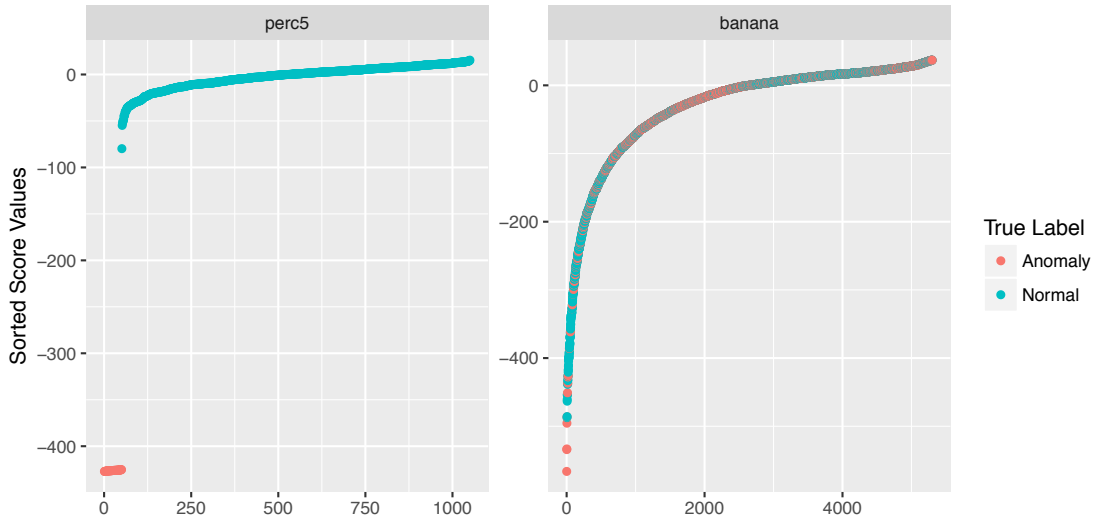


Figure 2: Sorted score values generated by an OC-SVM on the synthetic data set *5perc* and the *banana* data set (see Section 8.1).
Note: In unsupervised learning the observations usually can not be colored as there are no labels.

AUMVC is developed as measure to enable tuning in the unsupervised setting and is based on the Mass Volume curve. The theory of the MV curve for scoring anomalies was introduced by Clémençon and Jakubowicz [2013]. Based on this theory, Thomas et al. [2016] developed a method to assess the performance of scoring functions by extracting the MV curve and subsequently the AUMVC. The main difference between MV curve and AUMVC is that the MV curve assesses the performance of a scoring function at one point $\alpha \in [0, 1)$ (probability mass of the normal data) and the AUMVC within a certain interval $I = [\alpha_1, \alpha_2] \subseteq [0, 1)$.
The theory is further explained in the following subsections.
Overview of some notations in this chapter:

- $X = \{x_1, x_2, ..., x_n\}$: set of N observations drawn from the d-dimensional space $\mathbb{R}^d$, also called random variables
- $X = X_{train} \cup X_{test}$ the available observations are randomly split
- f/F: true underlying density/distribution of X in training and test set
- $\theta \in \Theta$ hyper parameter settings of the anomaly detection algorithm
- $\mathcal{A} : X \times \Theta \to \mathbb{R}^{\mathbb{R}^d}$ anomaly detection algorithm
- $(X \times \theta) \mapsto \mathcal{A}(X \times \theta) =: \hat{s}_{X,\theta}$ estimated scoring function based on the anomaly detection algorithm $\mathcal{A}$, the available data set X and the hyper parameter settings θ

- $S = \{s_1, s_2, ..., s_n\}$ score values for each observation in X that result from applying the scoring function $\hat{s}_{X,\theta}$; within this section low scores are considered to be an indicator for an anomalous observation.
- $\alpha \in [0, 1)$ the probability mass of the normal data
- $t \in \mathbb{R}$ is a threshold. If the score value of an observation is greater than the threshold it is assumed to be a normal observation, otherwise anomalous (i.e. low score values indicative of anomalies)
- $n_{sim} \in \mathbb{N}$ number of Monte-Carlo samples used to calculate $AUMVC$
- $d' \leq 8$ parameter controlling the number of features for feature sub-sampling, on which to apply AUMVC
- m number of subsamples with dimension $n_{sim} \times d'$ used to calculate $AUMVChd$
- $k = 1, ..., m$ indicate the k-th subsample

3.2.1 Mass-Volume Curve (MV curve)

To understand the idea behind the *MV curve for a scoring function* we constructed the MV curve step by step by using score values extracted from the synthetic data *5perc* and the *banana* data set (see Figure 2 and Section 8.1 for more details on the data set). The score values are calculated based on an OC-SVM with radial basis kernel and the default values from the R-package *e1071* [Meyer et al., 2015].
First step is to calculate the score values from the anomaly detection algorithm $\mathcal{A}$

$$\mathcal{A} : X \times \Theta \to \mathbb{R}^{\mathbb{R}^d} \tag{1}$$

with the scoring function $\hat{s}_{X,\theta}$ using the hyper parameter settings $\theta \in \Theta$ of the anomaly detection algorithm and the available data set X

$$(X \times \theta) \mapsto \mathcal{A}(X \times \theta) =: \hat{s}_{X,\theta} \tag{2}$$

If the estimated score function $\hat{s}_{X,\theta}$ is an increasing transformation of the true density f, the sorted score values can help to set a threshold. However, the scoring function strongly depends on the anomaly detection algorithm $\mathcal{A}$ and especially on the used hyper parameter setting θ. An example, where the estimated score function is not an increasing transformation of the true density f is shown for the pen.local data in Figure 12. If the score function is an increasing transformation of the true f, anomalies would lie in the tail of the true distribution.

To understand the intuition behind the Mass Volume curve we will disassemble its definition into their individual parts and plot them. As the name *"Mass Volume for a scoring function"* (3) indicates, the measure has two components: the *mass*, which is

defined as $\alpha_{s_{X,\theta}}$ (4) and the *volume*, which is defined as $\lambda_{s_{X,\theta}}$ (5) for a scoring function $s_{X,\theta}$. Furthermore, the MV curve is a parametric function of the threshold t.

$$MVC_{s_{X,\theta}} : t \in R \mapsto (\alpha_{s_{X,\theta}}(t), \lambda_{s_{X,\theta}}(t)) \tag{3}$$

with the mass at threshold t

$$\alpha_{s_{X,\theta}}(t) = \mathbb{P}(s_{X,\theta}(X) \geq t) \tag{4}$$

and the volume (w.r.t. Lebesgue measure) at threshold t

$$\lambda_{s_{X,\theta}}(t) = \lambda(x, s_{X,\theta}(X) \geq t) \tag{5}$$

For calculating the mass $\alpha_{s_{X,\theta}(t)}(t)$ the probability distribution $\mathbb{P}$ is needed, but as it is unknown Clémençon and Jakubowicz [2013, p.2], an empirical probability $\hat{\alpha}_{s_{X,\theta}}(t)$ is defined

$$\hat{\alpha}_{s_{X,\theta}}(t) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}_{\{x, s_{X,\theta}(x) \geq t\}}(x_i). \tag{6}$$

Figure 3 visualizes the behaviour of the empirical mass/probability $\hat{\alpha}_{s_{X,\theta}}(t)$ depending on the threshold t with values that lie in the range of the score values S.

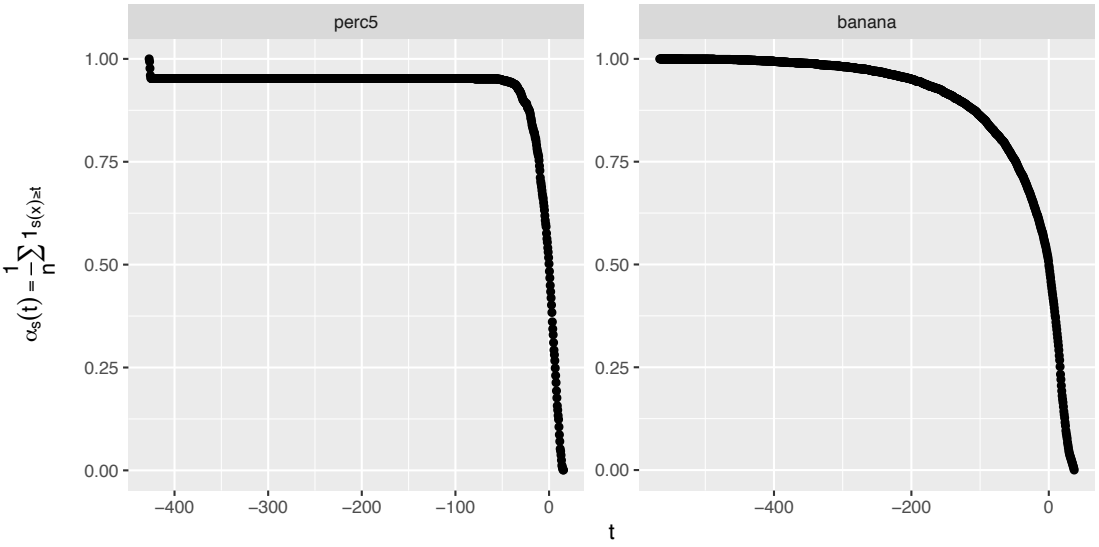


Figure 3: Empirical mass/probabilities $\hat{\alpha}_{s_{X,\theta}}$, which shows the relative frequency that the score values are greater than the threshold t. The score values are extracted with the scoring function $s_{X,\theta}$ generated by OC-SVM on the synthetic data set *5perc* (left) and the *banana* data set (right) (see Section 8.1 for more information on the data sets).

Figure 3 shows that with increasing thresholds t, the number of score values which are greater than the threshold t (6) are decreasing (less observations classified as normal). So the lower the threshold t the greater is the *mass* of the data classified as normal. In the left panel of the graphic (*5perc* data set) one can see that the threshold between -250 and -25 would yield the same mass $\alpha_{s_{X,\theta}}(t)$, whereas the right panel of the graphic (*banana* data set) shows that the *mass* varies with a varying threshold.
Figure 4 visualizes the behaviour of $\lambda_{s_{X,\theta}}(t)$ depending on the threshold t with values that lie in the range of the score values S.

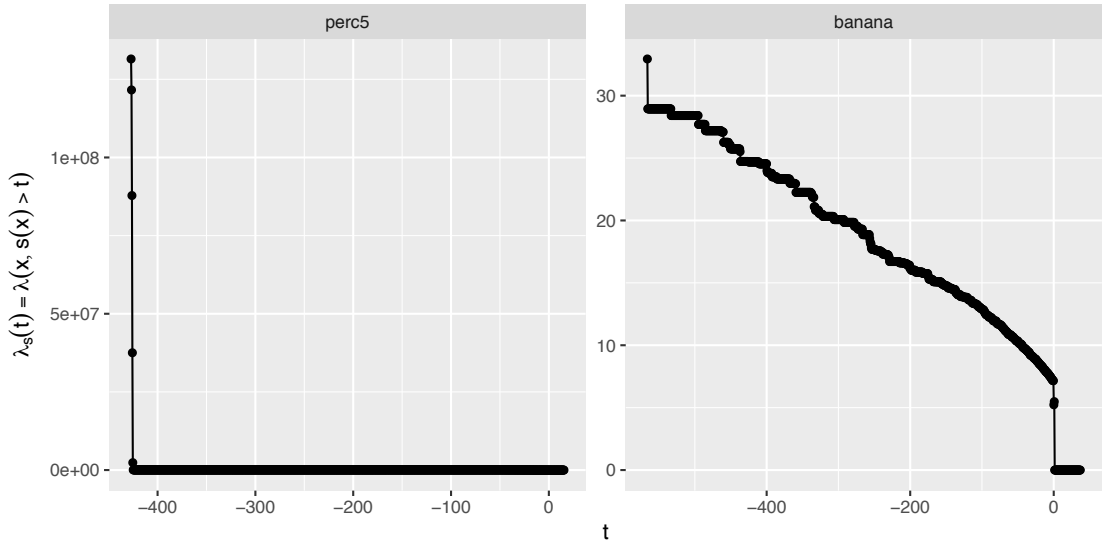


Figure 4: Lebesgue measure or volume $\lambda_{s_{X,\theta}}$ of the subset of x values for which the corresponding score value s_i is greater than the threshold t. The score values are extracted with the scoring function $s_{X,\theta}$ generated by OC-SVM on the synthetic data set *5perc* (left) and the *banana* data set (right); (see Section 8.1 for more information on the data sets).

Figure 4 shows, that with increasing threshold t the volume $\lambda_{s_{X,\theta}}$ (5) decreases, as less score values exceed threshold t. Therefore, the set $\{x, s_{X,\theta}(X) \geq t\}$ for calculating the volume is smaller.
Thus, with increasing threshold t, both, the *mass* and the *volume* in the Mass Volume curve definition are decreasing.
The definition of the MV curve up to this point has a two-dimensional output. If the mass $\alpha_{s_{X,\theta}}(t)$ has no flat parts, which means that its derivatives are non-zero at a given point [Glaister, 1991], than the MV curve can be defined as

$$MVC_{s_{X,\theta}} : \alpha \in [0,1) \mapsto \lambda_{s_{X,\theta}}(\alpha^{-1}_{s_{X,\theta}}(\alpha)) \tag{7}$$

where the inverse of $\alpha_{s_{X,\theta}}(t)$ is defined as

$$\alpha^{-1}_{s_{X,\theta}}(\alpha) = \inf\{t \in \mathbb{R}, \alpha_{s_{X,\theta}}(t) \leq \alpha\} \tag{8}$$

The curve of $\alpha^{-1}_{s_{X,\theta}}(\alpha)$ is shown in Figure 5.

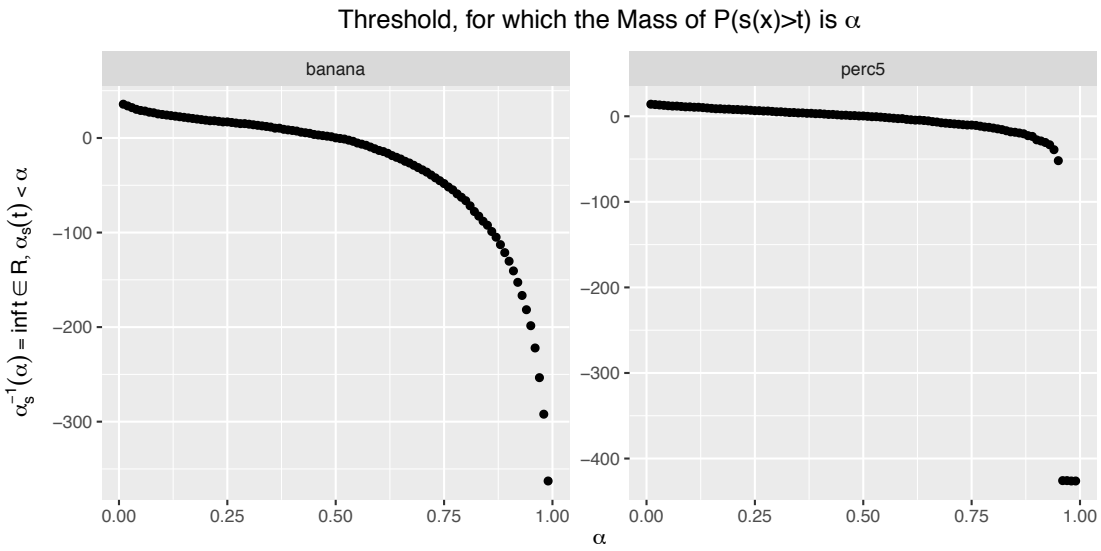


Figure 5: The inverse function of $\alpha_{s_{X,\theta}}(t)$ is $\alpha^{-1}_{s_{X,\theta}}(t)$ and is based on score values, which are extracted with the scoring function $s_{X,\theta}$ generated by OC-SVM on the synthetic data set *5perc* (left) and the *banana* data set (right); (see Section 8.1 for more information on the data sets).

As stated above the true density f is unknown, therefore we need to define the empirical MV curve of a scoring function, given X and the empirical $\hat{\alpha}_{s_{X,\theta}}(t)$:

$$\widehat{MVC}_{s_{X,\theta}} : \alpha \in [0,1) \mapsto \lambda_{s_{X,\theta}}(\hat{\alpha}^{-1}_{s_{X,\theta}}(\alpha)) \tag{9}$$

The MV curve with a one-dimensional output can be visualized as in Figure 6.

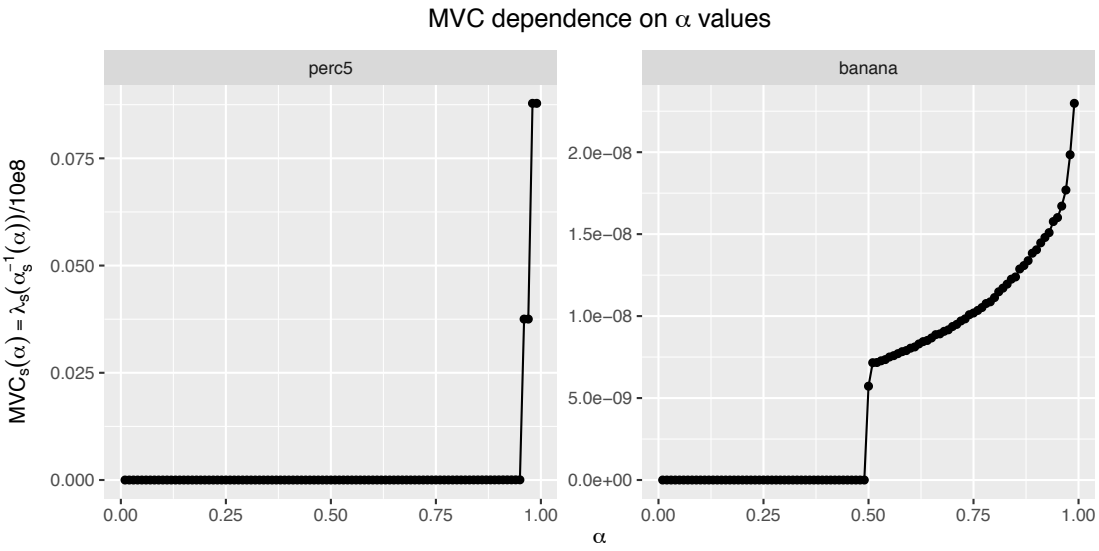


Figure 6: The $MV_{s_{X,\theta}(\alpha)}$ based on score values that are extracted with an OC-SVM from the synthetic data set *5perc* (left) and the *banana* data set (right); (see Section 8.1 for more information on the data sets).

Figure 6 shows, the higher the mass α of the normal data, the larger is the value of the MV curve.

The output of the $MVC_{s_{X,\theta}}$ characterizes the underlying score function $s_{X,\theta}$ from an anomaly detection algorithm at a given mass α i.e. it returns a performance measure for $s_{X,\theta}$ when the mass of the normal data is assumed to be α.
Moreover the MV curve can be used to compare different score functions and anomaly detection algorithms at a given mass point α. Optimal scoring functions have MV curves that are minimal everywhere according to Clémençon and Jakubowicz [2013, p.4]. In the paper it is stated that if s_1 and s_2 are two scoring functions on X, than the ordering provided by s_1 is better than that provided by s_2 when

$$\forall \alpha \in [0,1)\ MVC_{s_1}(\alpha) \leq MVC_{s_2}(\alpha) \tag{10}$$

Furthermore Clémençon and Jakubowicz [2013, p.4] provide a proposition about the optimal MV curve, which is the MVC_f curve based on the true density $f(x)$ of the random variable X. As the true density $f(x)$ is not known, the optimal MC curve should be based on the optimal scoring function, that ranks the observations X in the same order as $f(X)$, i.e., a strictly increasing transformation of $f(X)$. The set of optimal scoring functions is then defined as

$$S^* = \{T \circ f :\ T : Imf(X) \to \mathbb{R}_+, \text{strictly increasing}\} \tag{11}$$

The elements of S^*, therefore have the same MV curve and the same ordering as f.

Thus, the best possible ordering in regard to the MV curve criterion, which is defined as

$$\forall (s, \alpha) \in S \times [0, 1), \ MVC^*(\alpha) \leq MVC_s(\alpha) \tag{12}$$

$$\text{where } MVC^*(\alpha) = MVC_f(\alpha) \ \forall \alpha \in [0, 1) \tag{13}$$

So in the best case scenario $MVC_{s_{X,\theta}}(\alpha)$ equals $MVC^*(\alpha)$ for all α.

3.2.2 Area under the Mass Volume Curve (AUMVC)

So far the MV curve assesses the performance of a scoring function at one point α, the AUMVC assesses the performance of a scoring function over an interval $I = [\alpha_1, \alpha_2]$. The basic idea is to calculate the distance between the optimal Mass Volume Curve MVC^* and the estimated $MVC_{\hat{s}_{X,\theta}}$ using the L_1 distance over the interval I [Thomas et al., 2016, p.2]:

$$||MVC_{\hat{s}_{X,\theta}} - MVC^*||_{L_1} = \int_I |MVC_{\hat{s}_{X,\theta}}(\alpha) - MVC^*(\alpha)| d\alpha. \tag{14}$$

The distance indicates how far away the MV curve of the scoring function $\hat{s}_{X,\theta}$ is from the optimal MV curve, therefore the smaller the distance the closer is the estimation to the optimal curve. However, the distance can't be calculated directly, as the optimal Mass-Volume Curve MVC^* and the true density $f(x)$ are unknown. But the main objective is not to get the actual distance value, but to find the minimal distance to identify the optimal scoring function. Therefore, the optimal hyper parameter setting θ is defined by

$$\theta^* = \arg\min_{\theta \in \Theta} ||MVC_{\hat{s}_{X,\theta}} - MVC^*||_{L_1} \tag{15}$$

Equation (15) can be further simplified using equations (12) and (13). These equations say that the optimal Mass Volume Curve MVC^* is minimal for all $(s, \alpha) \in S \times [0, 1)$. Therefore minimizing the distance to MVC^* is equal to minimizing $MVC_{\hat{s}_{X,\theta}}$ over I.

$$\theta^* = \arg\min_{\theta \in \Theta} ||MVC_{\hat{s}_{X,\theta}} - MVC^*||_{L_1} = \arg\min_{\theta \in \Theta} ||MVC_{\hat{s}_{X,\theta}}||_{L_1} \tag{16}$$

Calculating $||MVC_{\hat{s}_{X,\theta}}||_{L_1}$ returns the Area Under the Mass Volume curve (AUMVC) of a scoring function:

$$AUMVC_I(\hat{s}_{X,\theta}) := ||MVC_{\hat{s}_{X,\theta}}||_{L_1} = \int_I MVC_{\hat{s}_{X,\theta}}(\alpha) d\alpha \tag{17}$$

To calculate the AUMVC in practice we need to consider the empirical MV curve (9), as the true MV curve is unknown [Thomas et al., 2016, p.2]. Combining (16) and (17),

the optimal hyper parameter set θ by minimizing AUMVC such that

$$\theta^* = \arg\min_{\theta \in \Theta} AUMVC_I(\hat{s}_{X,\theta}) \tag{18}$$

3.2.2.1 Tuning Hyper parameters with AUMVC

With the AUMVC we have a tool to compare two scoring functions, in the sense of which scoring function is the closer representative of the true density $f(x)$. The closer the scoring function is to the optimal density, the smaller is its AUMVC value. Therefore, the AUMVC provides a suitable measure for tuning in an unsupervised setting.
In unsupervised tuning, the usual tuning approach is to split the available data set into training set X_{train} and test set X_{test} to prevent overfitting [Thomas et al., 2016, p.2]. For a parameter set $\theta \in \Theta$ and an anomaly detection algorithm $\mathcal{A}$ we use X_{train} to determine the scoring function $\hat{s}_{X_{train},\theta}$ (see equations (1) and (2)). The performance of $\mathcal{A}$ is assessed by estimating the empirical AUMVC from the test set X_{test} on an interval $I = [\alpha_1, \alpha_2]$ [Thomas et al., 2016, p.2], that can be set by the user (see Section 3.2.2.4 on how to choose the interval). Applying (18) we therefore choose the parameter set θ^* as follows

$$\theta^* = \arg\min_{\theta \in \Theta} \widehat{AUMVC}_I^{test}(\hat{s}_{X_{train},\theta)} = \arg\min_{\theta \in \Theta} \int_{\alpha_1}^{\alpha_2} \widehat{MVC}^{test}_{\hat{s}_{X_{train},\theta}}(\alpha) d\alpha \tag{19}$$

$$= \arg\min_{\theta \in \Theta} \int_{\alpha_1}^{\alpha_2} \lambda_{\hat{s}_{X_{train},\theta}}(\hat{\alpha}^{-1}_{\hat{s}_{X_{train}}}(\alpha)) d\alpha \tag{20}$$

As integrating over the empirical MV curve is not obvious, Thomas et al. [2016, p.3] introduce an automatic tuning algorithm (see Algorithm 1), which is the core component for this tuning approach.

3.2.2.2 Computing the Volume in AUMVC

When running Algorithm 1, we need to calculate the volume $\lambda^{test}_{\hat{s}_{X_{train},\theta}}(\rho^{test}_{\beta})$ in a high dimensional space, which is known to be difficult [Thomas et al., 2016, p.3]. Therefore, we use the Monte-Carlo (MC) integration approach as a workaround. Applying the MC integration to the AUMVC calculation yields Algorithm 2.

3.2.2.3 AUMVC on High-Dimensional Data ($AUMVC_{hd}$)

Due to the curse of dimensionality it is only recommended for small dimensional data of up to eight dimensions [Goix, 2016, p.3]. In Thomas et al. [2016] and [Albert], the boundary for the dimension of the data is set to five dimension, in Goix [2016] it is set to eight dimension. Within this work we will take eight dimension as the maximal dimension for the AUMVC.
[Goix, 2016] introduce a solution to how to apply AUMVC on high dimensional data.

Algorithm 1: Automatic Hyper Parameter Selection

Require: $\mathcal{A}$ anomaly detection algorithm
X data set
X_{train} and X_{test} random split of X to a training and test set
Θ hyper parameter subspace
$[\alpha_1, \alpha_2]$ integration interval
n_α discretization parameter
for each hyper parameter $\theta \in \Theta$ **do**
  $\hat{s}_{X_{train},\theta} = \mathcal{A}(X_{train}, \theta)$
  for β in $\{\alpha_1 + j\frac{\alpha_2-\alpha_1}{n_\alpha-1}; j \in \{0, ..., n_\alpha - 1\}\}$ **do**
    Compute the offset ρ_β^{test},which is the empirical $1-\beta$ quantile of $\hat{s}_{(X_{train},\theta)}(X_{test})$ and $\lambda^{test}_{\hat{s}_{X_{train},\theta}}(\rho_\beta^{test})$ on the test data X_{test}
  end for
  Compute $\widehat{AUMVC}_I^{test}(\hat{s}_{X_{train},\theta})$
end for
return $\theta^+ = \arg\min_{\theta\in\Theta} \widehat{AUMVC}_I^{test}(\hat{s}_{X_{train},\theta})$ and $\hat{s}^+_{X_{train},\theta^+}$

Algorithm 2: Monte Carlo Integration in AUMVC for $\lambda^{test}_{\hat{s}_{X_{train},\theta}}(\rho_\beta^{test})$

Require: X_{test} test set
$\hat{s}_{X_{train},\theta}$ scoring function
n_{sim} number of samples to draw from the hypercube
ρ_β^{test} offset/threshold
  Calculate the hypercube of X_{test}:
    $H = \{(min(x_{test,1}), max(x_{test,1})), ..., (min(x_{test,d}), max(x_{test,d}))\}$
  Calculate the volume of the hypercube H:
    $V = \prod_{j=1}^{d}(max(x_{test,j}) - min(x_{test,j}))$
  Sample $n_{sim} \in [1, n]$ uniform samples from the hypercube H: U
  Calculate the score values of samples U: $S_U = \hat{s}_{(X_{train},\theta)}(U)$
  Calculate the relative ratio of normal classified observations in U:
    $r = \hat{\alpha}_{\hat{s}_{X_{train},\theta}}(\rho_\beta^{test}) = \frac{1}{n_{sim}} \sum_{i=1}^{n_{sim}} 1_{\{u_i, \hat{s}_{X_{train},\theta}(u) \geq \rho_\beta^{test}\}}(U)$
return the Volume of normal classified observations in U:
    $\lambda^{test}_{\hat{s}_{X_{train},\theta}}(\rho_\beta^{test}) = r * V$

The basic idea is to do several feature subsamples of dimension $d' \leq 8$ to reduce the dimension of the data. The number of subsamples are $m \in \mathbb{N}$ and $k = 1, ..., m$ indicate the k-th subsample, therefore AUMVC can be applied on each subsample, yielding partial scores $\widehat{AUMVC}_{(I,k)}$. The mean of the partial scores is the new performance criterion $\widehat{AUMVC}_{hd_I}$. A pseudocode based on Goix [2016, p.3] is displayes in Algorithm 3.
Drawback of Goix [2016] approach is, that the provided algorithm doesn't evaluate

Algorithm 3: Evaluate AUMVC algorithm on high-dimensional data

Require: Same input as Algorithm 1, with $X = (x_i^j)_{1 \leq u \leq n,\ 1 \leq j \leq d}$
$d' \in (1, d)$ and $d' \leq 8$ feature sub-sampling size
m number of draws
for $k = 1, ..., m$ **do**
randomly select subgroup F_k of d' features
the reduced data set is: $X^{d'} = (x_i^l)_{1 \leq i \leq n,\ l \in F_k}$
the reduced data train and test sets are: $X_{train}^{d'}$ and $X_{test}^{d'}$
compute the associated scoring function: $\hat{s}_{X_{train}^{d'}, \theta}$
compute the partial score $\widehat{AUMVC}_{(I,k)}^{test}(\hat{s}_{X_{train}^{d'}, \theta})$ on the test set according to Algorithm 1
end for
return aggregated performance criterion: $\widehat{AUMVC}_{hd_I} = \frac{1}{m} \sum_{k=1}^{m} \widehat{AUMVC}_{(I,k)}$

combinations of more than d' features Goix [2016]. However, experiments in Goix [2016] has shown that it is enough in most of the cases.

3.2.2.4 Choosing the Interval I for $AUMVC$ and $AUMVC_{hd}$

For calculating $\widehat{AUMVC}_I$ and $\widehat{AUMVC}_{(I,k)}$ we need to take the integral of the empirical MV curve over the interval $I = [\alpha_1, \alpha_2] \subseteq [0, 1)$. The upper bound is excluded, as the MV curve diverges in 1, if the support of the underlying distribution F is finite [Goix, 2016, p.2]. Furthermore, in anomaly detection, one is only interested in outliers (the extreme values) and assumes that it is a rare event. Therefore, α_1 and α_2 should be large values [Clémençon and Jakubowicz, 2013, p.4], e.g. $[0.9, 0.99]$ [Thomas et al., 2016, p.2]. Because choosing e.g. $\alpha = 0.9$ means that we want to estimtate the MV curve, where the *mass* of normal classified data is 90%, in classical anomaly detection it is not reasonable to assume to have more than 10% anomalies.

In Thomas et al. [2016] and Goix [2016] it is shown that the $AUMVC_I$ and $AUMVC_{hd_I}$ do not require labeled data to evaluate an anomaly detection algorithm. Thomas et al. [2016] compare performance of anomaly detection algorithms with fixed hyper parameters (set accordingly based on theoretical results or results of the the original authors) to performance of anomaly detection algorithms with tuned parameters based on Algorithm 1. The performance of both settings is assessed with $AUMVC$ and tuning the anomaly

detection algorithm with $AUMVC$ always performs better than using prior fixed hyper parameters according to [Thomas et al., 2016, p.4]. In our experiments (Section 9.2) the results do not alway support that statement. Goix [2016, p.2] also compares anomaly detection algorithms using the ROC, PR and $AUMVC$ or $AUMVC_{hd}$ as performance measures. His experiments show that in 72% to 76% of his examples the $AUMVC$ or $AUMVC_{hd}$ recovers the order of ranking [Goix, 2016, p.24].

Note: The default predicted probability in mlR is of the anomaly class, anonamous observations should have high probabilities. Therefore the probability of the normal class is used to calculate the AUMVC(hd).

4 Unsupervised Anomaly Detection Methods

A variety of anomaly detection methods exists for unsupervised learning, for example, based on support vector machines (e.g. one-class classification SVMs in Section 4.1), based on density (e.g. LOF in Section 4.2) or neural networks (autoencoder in Section 4.3). Furthermore, there are more method types and methods, such as cluster-based methods or k-nearest neighbors to name a few, which are not part of this work. Anomaly detection methods can differ in their requirement of data labels as mentioned in Section 2. For instance, one-class SVMs (OC-SVMs) and autoencoders requires a semi-supervised setting (one class in training), whereas LOF is defined for the unsupervised setting.
Within this work, we will focus on the three previously mentioned methods, which are introduced in more detail in the following section and OC-SVM and LOF are also benchmarked in Section 8.

4.1 Support Vector Machine (SVM)

This section provides a brief concept of SVM and the separating hyperplanes [James et al., 2013, p.343], a very popular machine learning method [James et al., 2013, p.360]. SVMs were originally developed for binary classification problems [James et al., 2013, p.341]. However, SVMs can be extended to the case with more than two classes or to the case of regression, which is not in the scope of this work (for more details see [James et al., 2013, chapter 9.4]). SVMs can also be used for anomaly detection or also called one-class classification, which will be explained in more details in Section 4.1.2.
The development of SVMs is quite intuitive explainable on the example of binary classification. A first overview and summary of [James et al., 2013, chapter 9] is displayed in Figure 7 and Table 1. Figure 8 explains visually the main terminology. In this section, we quickly step through each method in the knots of Figure 7, giving a more but short explanation about usages and limitations. For more details and derivation of used equations see [James et al., 2013] and [Friedman et al., 2001]. For all equations in this section following notation holds:

- $x_i \in \mathbb{R},\ i = 1, ..., n$ and $n \in \mathbb{N}$ are elements of the data D.

- $y_i \in -1, 1,\ i = 1, ..., n$ are the binary class labels for x_i. In an unsupervised setting there are no class labels available.
- $\phi : D \to F$ is the feature map, which maps into the product space F
- $\omega \in F$ element of the product space F
- $k(\cdot, \cdot) = \langle \cdot, \cdot \rangle : D^2 \to \mathbb{R}$ is a kernel function. For a linear kernel, it equals the inner product
- $\xi \in \mathbb{R}^n$ is the slack variable (one for each observation)
- $\nu \in (0, 1)$ controls the number or fraction of anomalies to be found
- $p \in \mathbb{R}$ is the offset

If you already familiar with the general motivation of SVMs, you can proceed with Section 4.1.1 about the kernel trick or Section 4.1.2 about one-class SVMs.

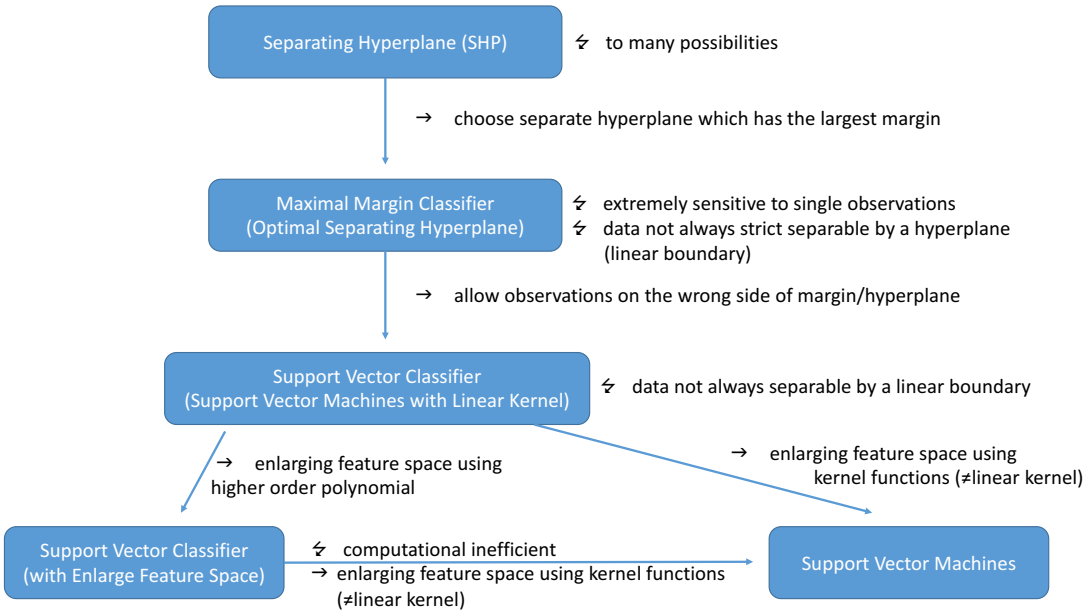


Figure 7: Several possible classification approaches based on concept of separating hyperplanes and motivation for introducing Support Vector Machines. Extraction based on [James et al., 2013, chapter 9].

Separating Hyperplane: Basically, the idea is to separate two classes by drawing a linear border between them. In a two-dimensional space the border would be a line (see Figure 8), in a three-dimensional space the border would be a plane and in any dimension, the border is, in general, called a separating hyperplane, which is linear in

Table 1: Overview of applicability of methods based on the concept of separating hyperplanes for different type of data sets. Extraction based on [James et al., 2013, chapter 9].

	Data Set Strict Separable by Linear Boundary (Strict Separating Hyperplane Exists)	Data Set Separable by Linear Boundary, But Not Strict (Hyperplane Exists)	data sets with Non-Linear Class Boundaries
Maximal Margin Classifier	✓	x	x
Support Vector Classifier	✓	✓	✓
Support Vector Machine	✓	✓	✓

the corresponding space. But in Figure 8 one can easily see, that there are several options to set the separating hyperplane w, any w which hold following equation

$$y_i(\langle \omega^T, x_i \rangle + \rho) \geq 0 \tag{21}$$

with $\langle \cdot, \cdot \rangle$ the inner product.
At first sight, it seems like it doesn't matter, as it separates the two classes in the right way, but if we want to classify new instances, which additionally gather near the separating hyperplane, the exact position of the separating hyperplane does matter.

Maximal Margin Classifier: To set the optimal separating hyperplane, one wants to yield maximal margin. In other words, we want to choose the separating hyperplane, which has a maximal distance to the nearest observations or "that has the farthest minimum distance to the training observations" [James et al., 2013, p.345]. The equation (21) is extended to

$$\min_{\omega,\rho} \frac{1}{2}\omega^T\omega \tag{22}$$

$$\text{subject to } y_i(\langle \omega^T, x_i \rangle + \rho) \geq 1 \tag{23}$$

The separating hyperplane is then called the maximal margin hyperplane, which is also known as the maximal margin classifier because the observation can be classified depending on which side of the hyperplane the observation lies. The observations, which determine the margin, are called support vectors. But there is a limitation of this approach: a slight change of the support vectors leads to a change of the position of the separating hyperplane and in many cases no separating hyperplane exists.

Support Vector Classifier: To fix the limitation of the maximal margin classifier we allow the separating hyperplane to *only almost* separate the classes. Which basically means, that we allow observations to be on the wrong side of the hyperplane or even wrong side of the margin, by using a *soft margin* and introducing the slack variable ξ. ξ penalizes if the observation i lies on wrong side of decision boundary. This leads to a

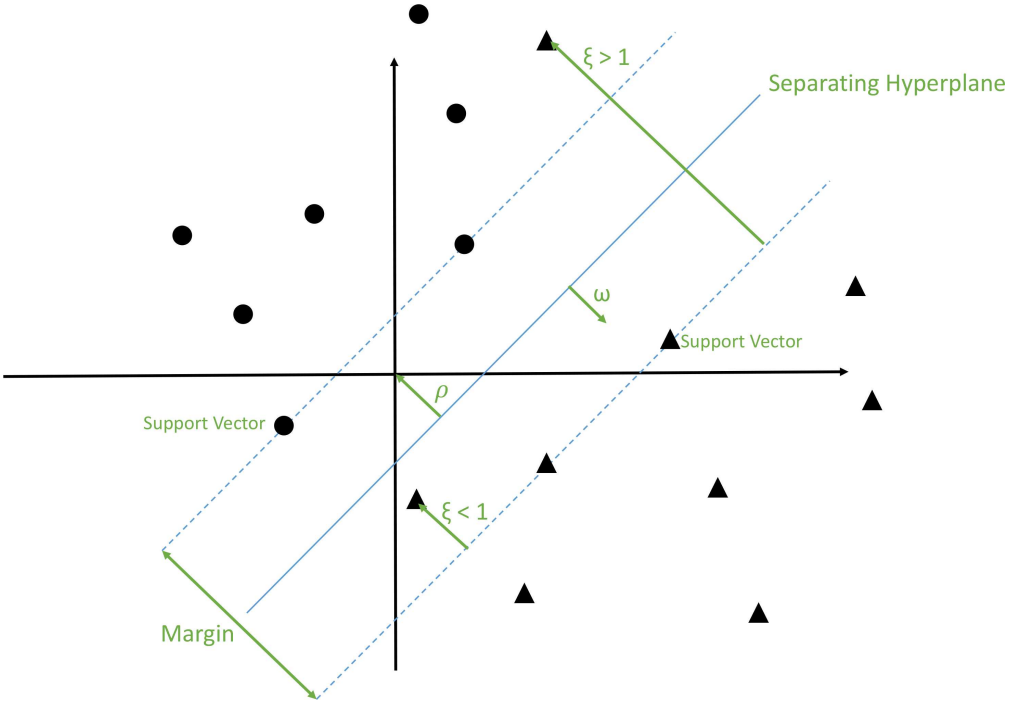


Figure 8: Visual definition of main terminology used for explaining SVM. (Später austauschen mit eigener grafik)

more robust behaviour to change in individual observations:

$$\min_{\omega,\xi\rho} \frac{1}{2}\omega^T\omega + C\sum_{i=1}^{n}\xi_i \tag{24}$$

$$\text{subject to } y_i(\langle\omega^T, x_i\rangle + \rho) \geq 1 - \xi_i \tag{25}$$

$$\xi_i \geq 0,\ i = 1, ..., n \tag{26}$$

where C is a trade-off variable, controlling the relative importance between the two objectives in (24).
The maximal margin classifier in the combination of the soft margin is called the *support vector classifier*. However, until now we assume that the data is linearly separable by a hyperplane in the space in which the instances are observed, which is not always possible. But the instances are separable in a higher dimension, so a solution would be to enlarge the feature space by using higher order polynomial terms for the predictor's functions. The fitted support vector classifier would be linear in the enlarged feature space, but non-linear in the original feature space. There are many possibilities to enlarge the feature space, but this comes at the price of computability.

Support Vector Machines In the case of non-linear class boundaries the support vector classifier is not sufficient unless we enlarge the feature space by using e.g. polynomials for the predictor's functions, which also increases the computational costs. As the support vector classifier only uses the inner products of the observations to find a solution, we replace this calculation (of enlarging the feature space) with a generalization of the inner product, which is referred to as *kernel function*. Thus inner product $\langle \cdot, \cdot \rangle$ in (25) is replaced by the the kernel function, hence the support vector classifier becomes the support vector machine.

$$\min_{\omega,\xi\rho} \frac{1}{2}\omega^T\omega + C\sum_{i=1}^{n}\xi_i \tag{27}$$

$$\text{subject to } y_i(\langle \omega^T, \phi(x_i)\rangle + \rho) \geq 1 - \xi_i \tag{28}$$

$$\xi_i \geq 0,\ i = 1, ..., n \tag{29}$$

$\langle \cdot, \cdot \rangle$ is a kernel function chosen by the user (see kernels in Section 4.1.1) and x_i is a map into the dot product space F using the feature map $\phi()$. The dot products in F can be calculated by using kernels.
The SVM is about efficient computation of learning a linear decision boundary in a higher-dimensional space and finally projecting the learned linear decision boundary back to the original feature space, which then becomes non-linear.
There are different types of kernel functions, linear kernels and non-linear kernels, such as the radial basis kernel or polynomial kernel (for more details see Section 4.1.1). When combining the support vector classifier with a non-linear kernel, the resulting classifier is called the *support vector machine (SVM)* [James et al., 2013, chapter 9]. When using the linear kernel instead, the SVM is reduced to the support vector classifier. In summary, with SVM one can yield computationally efficient non-linear class boundaries.

4.1.1 Kernels and The Kernel-trick

Most of the time it is difficult to extract structure from data in the space in which the observations have been made. To overcome this, a kernel-based SVM maps the data into a high dimensional feature space using kernel functions, in which the model is then trained [Karatzoglou et al., 2004, p.1]. Formally, the input data X are mapped into a high dimensional feature space F using a function $\phi : X \to F,\ x \to \phi(x)$. Then the inner product of the modified input is calculated. Every kernel can be reformulated as a dot product, which means, the modified input can be compared using a dot product $k : X^2 \to \mathbb{R}$ [Jordan, p.3].

$$k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle \tag{30}$$

Technically, the mapping ϕ is implicitly done, it is not even necessary to know ϕ. We only need the kernel $k()$, that does the mapping and the inner product in one operation. The kernel trick is to compute the dot products in the higher dimensional feature space F but with vectors from the input space and a function that also worked in the input

space [Jordan, p.3]
Table 2 gives an overview and provides definitions of common kernels used in SVM. Note that the definitions differ between publications; the definitions here are from the R packages *e1071* Meyer et al. [2015] and *kernlab* Karatzoglou et al. [2004].

Table 2: Common Kernels, which are used for SVMs. In some definition γ is used instead of σ.

Kernel	Equation $k(x_i, x_j) =$	Parameters
Linear	$\boldsymbol{x}_i^T \boldsymbol{x}_j$	none
Polynomial	$\sigma(\boldsymbol{x}_i^T \boldsymbol{x}_j + c_0)^d$	σ, d, c_0, for some Definition $\sigma = 1$
Gaussian Radial Basis Fct.	$exp\{-\sigma \lvert \boldsymbol{x}_i - \boldsymbol{x}_j \rvert^2\}$	σ
Sigmoid or Hyperbolic Tangent Kernel	$\frac{Bessel_{(}^{n}\nu+1)(\{\sigma \lvert \boldsymbol{x}_i - \boldsymbol{x}_j^T \rvert)}{\lvert x_i - x_j \rvert^{-n(\nu+1)}}$	σ, c_0
Bessel	$tanh\{\sigma \boldsymbol{x}_i^T \boldsymbol{x}_j + c_0\}$	σ, c_0
Laplace Radial Basis Fct.	$exp\{-\sigma \lvert \boldsymbol{x}_i - \boldsymbol{x}_j \rvert\}$	σ
Anova Radial Basis Fct.	$\left(\sum_{k=1}^{n} exp - \sigma(\boldsymbol{x}_i^k - \boldsymbol{x}_j^k)^2\right)^2$	σ

The listed kernels also satisfy the requirements for kernel functions, that are

- $k()$ is a symmetric function
- $k()$ satisfies the inequalities (follows from the Cauchy-Schwarz inequality)
- $k()$ is a inner product kernel (follows from Mercer's Theorem)

For more details, see Scholkopf [2001].

4.1.2 Support Vector Machines for Anomaly Detection

Until now the SVM was introduced for supervised learning. For one-class classification, there are two types of OC-SVMs applicable to anomaly detection problems: one from Schölkopf et al. [2000] and one from Tax and Duin [2004]. The latter detects anomalies by creating a decision boundary, that encloses a set of data points like a sphere [Karatzoglou et al., 2004, p.9] and minimizes volume of the hypersphere to minimize the chance of accepting outliers [Tax and Duin, 2004, p.47]. This spherical decision boundary is described by a set of support vectors [Tax and Duin, 2004, p.59] [Karatzoglou et al., 2004, p.9]. Within this work, we will focus on the definition of Schölkopf et al. [2000] that uses a planar instead of a spherical approach. For the Gaussian kernel (see 4.1.1), the two definitions are equal [Tong and Svetnik, 2002, p.2].
The difference between binary classification with SVM and OC-SVM is that SVM separates two classes while having examples of both classes in the training data, therefore the classical SVM can find a line, plane or hyperplane to best separate the data (often using the kernel trick to find linear separation in a higher feature space). The OC-SVM, however, only has examples of one class, thus the OC-SVM tries to find a line, plane or hyperplane to separate all the observations of one class from the origin. For the benchmark analysis in this work, we focus on the Gaussian kernel like [Tong and Svetnik,

2002, p.4].
For the primal problem formulating of detecting anomalies according to Schölkopf et al. [2000] and [Chang and Lin, 2011, p.5] is

$$\min_{\omega,\xi\rho} \frac{1}{2}\omega^T\omega - \rho + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i \tag{31}$$

$$\text{subject to } \langle \omega^T, \phi(x_i)\rangle \geq \rho - \xi_i \tag{32}$$

$$\xi_i \geq 0,\ i = 1, ..., n \tag{33}$$

with $\nu \in (0, 1)$ controlling the number or fraction of anomalies that is expected in training. ξ is the slack variable, that penalized the objective function if it's positive. In other words it penalizes if an observation lies on the wrong side of the decision boundary. Reformulation leads to the dual problem

$$\min_{\beta} \frac{1}{2}\beta^T Q\alpha \tag{34}$$

$$\text{subject to } 0 \leq \beta_i \leq 1/\nu n,\ i = 1, ..., n \tag{35}$$

$$\sum_i \beta_i = 1, \tag{36}$$

where $Q_{ij} = K(x_i, x_j) = \langle \phi(x_i)^T \phi(x_j)\rangle$. An observation x_i for which β_i is nonzero is called support vector (SV) [Schölkopf et al., 2001, p.583]. The decision function for detecting an observation as anomaly

$$sgn(\sum_{i=1}^{n} \beta_i K(x_i, x) - \rho) \tag{37}$$

Equation (37) is a function that returns $+1$ if it is a normal observation and -1 if it is an anomaly [Schölkopf et al., 2000, p.1].
Advantages of the one-class SVMs are

- SVMs have convex optimization objectives ensuring that the global optimum will be reached [Amer et al., 2013, p.2]
- SVMs have sparse solutions, therefore they are efficient in comparison to other kernel-based approaches [Amer et al., 2013, p.2]
- SVMs for anomaly detection contains a decision function, therefore the method both returns an anomaly score and a binary decision for the normal and anomaly class
- SVMs consider the fact that data can be noisy, thus the algorithm allows observations to be on the wrong side of the margin [Bischl, 2016]

- Infinitely many kernels exist, and many of them are efficiently computable [Bischl, 2016]

There are also a view disadvantages such as

- The method is not applicable for all data sets as it is requires that the classes are separable by a linear boundary [James et al., 2013, p.341]
- SVMs scale not well to increasing data sizes, because of the kernel transformation and the quadratic optimization algorithm [Meyer et al., 2015, p.8]
- The results of SVM models highly depend on the choice of the kernels, therefore an extensive parameter search is required [Meyer et al., 2015, p.8]
- The choice of a good kernel and its parameters is the responsibility of the user and takes a lot of effort to determine those [Bischl, 2016]

4.1.3 Tuning

For the OC-SVM, there is also a set of hyperparameters that needs to be defined before applying the method. However, it is not obvious to choose a hyper parameter setting and it highly depends on the data and the domain of the problem. One way to find this optimal set is to tune the hyperparameters.
There are parameters that are part of the definition of the method, and parameters that control computation time. Both have an impact on the result. The former for obvious reason, the latter in the sense that if, e.g., the number of iterations is limited, the algorithm might stop before it converged and therefore differs from the results that it would have yielded if it wouldn't stop prematurely. For OC-SVMs the parameter of the methods which can be tuned or set by the user according to the use case, are as follows:

- **kernel type**: To define the parameter space it is necessary to set a projection function that can be any kernel. The common kernels are listed in Table 2. In our benchmark analysis it is set to the Gaussian Kernel, therefore in the following only relevant parameters are listed.
- $\boldsymbol{\sigma}$: A parameter which needs to be set for the kernels (except linear kernels) is the σ parameter. It controls the degree of influence of the observations in the training set. The lower σ the stronger is the influence of each observation. The higher σ the weaker is the influence of each observation, thus it increase the smoothness or generalizability of the model.
- $\boldsymbol{\nu}$: $\nu \in (0, 1)$. For $\rho \neq 0$ in formula (33), ν can be seen as upper bound on the fraction of outliers and as a lower bound on the fraction of support vectors [Schölkopf et al., 2000, p.584]. The tuning parameter C, which determines the softness of the margin in the classification SVM is replaced by $\frac{1}{\nu l}$ in the OC-SVM.

- $\nu \to 0$: upper boundaries on the Lagrange multipliers in formula (36) $\frac{1}{\nu l} \to \infty$. Therefore the constraint $\sum_i \alpha_i$ becomes void [Schölkopf et al., 2000, p.584]. The penalization of errors in formula (32) becomes infinite, therefore Equation (32) can only be solved if the slack variable ξ_i is 0. For all i or in other words if the data is strictly linearly separable.
- $\nu \to 1$: upper boundaries on the Lagrange multipliers $\frac{1}{\nu l} \to \frac{1}{l}$. In order to meet the constraint $\sum_i \alpha_i = 1$, all errors need to be on the upper bound $\frac{1}{l} > 0$, which means that all x_i are support vectors.

Due to limitation of time and computational power, in our experiments we only use the Gaussian Kernel and tune two parameters (ν and σ) with a random search for the OC-SVM.

4.2 Local Outlier Factor (LOF)

The local outlier factor (LOF) is an unsupervised anomaly detection method that assigns to each observation a degree of being a local outlier [Breunig et al., 2000, p.1]. The degrees depend on the density of the neighborhood of each observation [Breunig et al., 2000, p.1]. Only a restricted number k of neighbors is taken into account for calculating the LOF value, which is why the method is *local*. The more isolated the observation is from its surrounding neighbors, the more likely it is to be an outlier. Therefore a high LOF indicates a high degree of *outlier-ness* Breunig et al. [2000, p.1].
The LOF method is related to density-based methods [Breunig et al., 2000, p.2] that typically have two hyperparameters: The number of neighbors k and the volume of the neighborhood ϵ. In the LOF method, k will be kept as a parameter, ϵ will be replaced dynamically (by $reach-dist_k(p)$, see defintion (41)), as the aim is to compare the densities of different sets of objects [Breunig et al., 2000, p.4]. Furthermore, in density-based methods the distance function can be chosen by the user, in LOF it is usually set to the Euclidean distance (like in [Breunig et al., 2000] or the `R` package `dbscan`), but the user could and should replace the distance function according to the use case.
[Breunig et al., 2000] introduces $LOF_k(p)$ in order to avoid only detecting global outliers as in other known outlier detection methods (e.g. DB-Outlier, see example in [Breunig et al., 2000, p.2]). For LOF several preceding definitions are needed:

k-Distance of an Object p k-distance of an object $p \in D$ is the distance $d(p, o)$ with $o \in D$, for which following requirements hold:

$$(i) \quad \text{for at least } k \text{ objects } o' \in D \setminus \{p\} \text{ it holds that } d(p, o') \leq d(p, o) \tag{38}$$

$$(ii) \quad \text{for at least } k-1 \text{ objects } o' \in D \setminus \{p\} \text{ it holds that } d(p, o') \leq d(p, o) \tag{39}$$

for $k \in \mathbb{N}_+$ and $d(p, o)$ the minimum distance between p and o, with both elements of the set D [Breunig et al., 2000, p.3]. The example in Figure 9 fulfills both requirements. The distances $d(p, o_1), d(p, o_2), d(p, o_3)$ are smaller or equal the distance $d(p, o)$ (requirement

(38)), the distances $d(p, o_1), d(p, o_2)$ are strictly smaller than $d(p, o)$ (requirement (39)), therefore $d(p, o)$ is the k-distance

k-Distance Neighbourhood of an Object p The k-distance neighbourhood of an object p for $k \in \mathbb{N}_+$ is defined as

$$N_{k-distance(p)}(p) = \{q \in D \setminus \{p\} | d(p, q) \leq k - distance(p)\} \tag{40}$$

$N_{k-distance(p)}(p)$ contains all the objects q, which are not further away from p than the $k - distance(p)$ and is also called the k-nearest neighbors of p. It should be emphasized that the set of k-nearest neighbors of p can have more than k elements when several objects have the same distance to p. [Breunig et al., 2000, p.3]

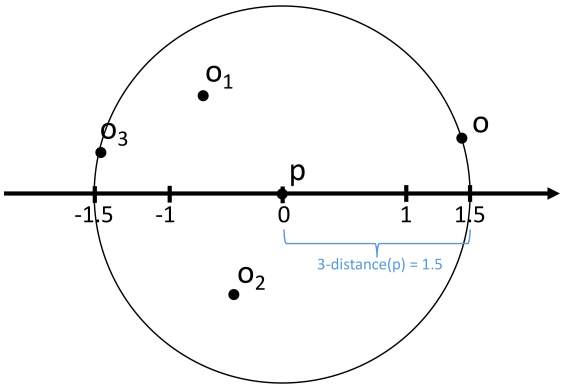


Figure 9: Two-dimensional, visual example of k-distance(p) and $N_{k-distance(p)}(p)$. Given $k = 3$, the $3 - distance(p)$, that holds (38) and (39) is 1.5. The 3-distance neighborhood or so called 3 nearest neighbors $N_{3-distance(p)}(p) = N_{1.5}(p) = \{o, o_1, o_2, o_3\}$.

Reachability Distance of an Object o w.r.t. object p The reachability distance of an object o w.r.t. object p for k in $\mathbb{N}_+$ is

$$reach - dist_k(o, p) = max\{k - distance(p), d(p, o)\} \tag{41}$$

The reachability distance of o to p is the true distance $d(p, o)$, but at least the k-distance of p. In other words, if o is far from p than the reachability distance is the true distance, and if o and p are "sufficiently" close, the reachability distance equals the k-distance of p. In the example in Figure 9 o_1 is too "close" to p, the distance $d(p, o_1)$ is less than the $3 - distance(p)$, therefore $3 - distance(p) = 1.5 = reach - dist_3(o_1, p)$ is the reachability distance of o_1 w.r.t. p.
The intention behind this definition is to reduce the statistical fluctuation of $d(p, o)$ for all objects o, that are close to p. k is a tuning parameter: The higher the value of k, the

more similar are the reachability distances of objects in the same neighborhood [Breunig et al., 2000, p.3].

Local Reachability Density of an Object p

$$lrd_k(p) = \frac{1}{\left\{ \frac{\sum_{o \in N_k(p)} reach\text{-}dist_k(p,o)}{|N_k(p)|} \right\}} \tag{42}$$

The equation returns the local reachability density of an object p. First all possible reachable distances of p to each of its k-neighbors are calculated, then averaged and inversed.
If the datasets have duplicates the nominator can be 0 and therefore $lrd_k(p)$ can be ∞. We follow the approach of [Breunig et al., 2000, p.4] and assume that there are no duplicates or, if duplicates are present, delete duplicates in the data. [Breunig et al., 2000, p.4]

Local Outlier Dactor of an Object p

$$LOF_k(p) = \frac{\sum_{o \in N_k(p)} \frac{lrd_k(o)}{lrd_k(p)}}{|N_k(p)|} \tag{43}$$

$LOF_k(p)$ returns the degree to which extent p is an outlier. First, the ratio of the local reachability of p and those of p's k-nearest neighbors (in $N_k(p)$) are calculated and then averaged. So the local reachability is compared between the object p (denominator) and o (numerator). A higher numerator in comparison to the denominator indicates that p lies in a sparser region than its k-neighbors. The higher $LOF_k(p)$, the more likely is p to be an outlier.
A LOF value close to one is a sure normal observation [Breunig et al., 2000, p.4], but it doesn't mean that a LOF-value above one is a sure outlier. [Breunig et al., 2000, p.4]

Advantages:

- LOF is well applicable on linearly separable distributions with stable densities [Lyudchik, 2016]
- Although LOF is related to density-based clustering, it does not require any explicit or implicit notion of clusters [Breunig et al., 2000, p.1]
- No information about the underlying distribution is necessary [Breunig et al., 2000, p.2]

Disadvantages:

- There are no rules for an outlier threshold for the LOF-values [Breunig et al., 2000, p.6]
- LOF only works with numeric data [Zhao, 2012, p.66]
- LOF tend to underperform on data with underlying nonlinear structures and variation in densities [Lyudchik, 2016]

4.2.1 Tuning

As previously explained, the only hyperparameter in the LOF method is the number of neighbors k. There is neither a rule of thumb how to choose k nor could it be done intuitively. [Breunig et al., 2000, p.7] show in their paper that there is no monotonic decrease or increase in LOF when varying k, instead it changes non-monotonically. [Breunig et al., 2000, p.2] also suggest to set k at least to ten in order do to remove statistical fluctuation that is unwanted [Breunig et al., 2000, p.8]. All in all, in their experiment a k between 10 and 20 works well in general as a lower bound. A further limitation on how to choose an upper or lower bound of k is described in [Breunig et al., 2000, p.8], which requires some information about the data. In this work we will focus on the general application of LOF on a variety of data, therefore we will tune the k parameter, without setting data-specific lower and upper bounds of k.

4.3 Neural Network (NN)

Neural networks (NN) are another class of learning methods for non-linear statistical models for regression and classification [Friedman et al., 2001, p.389]. First, the basic structure of a neural network is explained, followed by the introduction of a special neural network for anomaly detection.
A NN can be represented by a network diagram as shown in Figure 10. The features $X_1, X_2, ..., X_P$ are the P-input variables (input layer) and $Y_1, Y_2, ..., Y_K$ are the output variables (output layer). In classification, K indicates the number of classes and in regression, K is a scalar. The variables inbetween are called hidden neurons (in the hidden layers) of the neural network. The number of neurons and hidden layers can vary [Friedman et al., 2001, p.392]. The displayed network starts from left and runs to the right, without going back, feeding the input forward to the next layer, that is why it is called feed-forward neural network. Other architectures can be found in [Priddy and Keller, 2005]. Once the user has chosen an architecture of the NN, the network can start learning to map the input layer to the output layer.
The explanation of the structure of a NN is based on the exemplary neural network in Figure 10 and is a modification of [Friedman et al., 2001, p.389] explanations. To keep it simple, here we assume that every layer can only receive input from the closest previous layer and feed forward to the closest next layer. However, it should be noted that more complex architectures are possible.

In the following, the notation relevant to this this chapter is summarised:

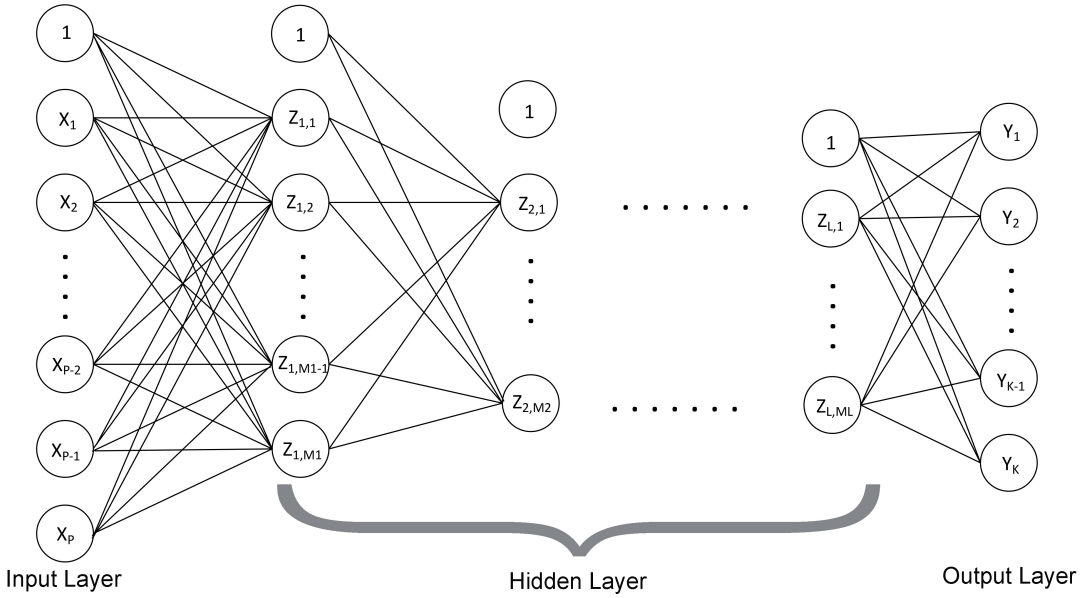


Figure 10: Schematic of a feed-forward neural network with at least three hidden layers, N input variables and a bias terms 1 in each layer and K output units in the output layer.

- $\{X_1, X_2, ..., X_P\}$ are the input in the input-layer. P defines the number of features. The layers each also has a bias term 1. Each X_p is a vector of N observations.
- Output $Y = \{Y_1, Y_2, ..., Y_K\}$ are output units in the output layer. In classification $K > 1$ defines the number of classes, in regression $K = 1$
- D is a NxP dimensional data set containg the input and output units.
- $l \in \{1, ..., L\}, L \in \mathbb{N}_+$ indicates the hidden layer.
- Hidden Neurons are denoted by $Z_{l,1}, Z_{l,2}, ...Z_{l,M_l}$, with M_l the number of hidden neurons in the l-th hidden layer.
- w_{jl,m_l}, $j \in \{0, 1, ..., M_{l-1}\}$, $l \in \{1, ..., L\}$, $m_l \in \{1, ..., M_l\}$ indicates the weight of the jth node from the previous layer $(l-1)$ for the m_l-th node of the lth layer. The weight for $j = 0$ belongs to the bias term, the weight for $l = o$ belongs to the output layer.
- $\sigma()$ is the activation function, that can be used for non-linear transformation
- $g()$ is the output transformation function
- $S = \{s_1, s_2, ..., s_N\}$ anomaly score values for each instance $\{1, ..., N\}$ in the data set

- t is the threshold of the anomaly scores

First Step: Input Layer to First Hidden Layer The second layer (first hidden layer) with hidden neurons $Z_{1,1}, Z_{1,2}, ...Z_{1,M_1}$ is created from linear combinations of the input variables (input layer) $X_1, X_2, ..., X_P$, such that

$$Z_{1,m_1} = \sigma(w_{0_{1,m_1}} + \sum_{i=1}^{P} w_{i_{1,m_1}} X_i),\ m_1 = 1, ..., M_1, \tag{44}$$

where $\sigma()$ is an activation function applied on each linear combination. The activation function is often chosen to be the sigmoid function $\sigma(\nu) = 1/(1 + e^{-\nu})$, defining which hidden neuron is "active" (> 0) or "inactive" ($= 0$).
Other most popular activation functions are the *tanh* function, rectified linear function and *maxout* function (see Table 3).
The first hidden layer passes the received and modified input variables forward to the next hidden layer.

Table 3: Activation functions [Candel et al., 2015, p.12].

Function	Equation	Range
Sigmoid	$\sigma(\nu) = 1/(1 + e^{-\nu})$	$\sigma(\cdot) \in [0, 1]$
Tanh	$\sigma(\nu) = \frac{e^{\nu} - e^{-\nu}}{e^{\nu} + e^{-\nu}}$	$\sigma(\cdot) \in [-1, 1]$
Rectified Linear	$\sigma(\nu) = \max(0, \nu)$	$\sigma(\cdot) \in \mathbb{R}_+$
Maxout	$\sigma(\nu_1, \nu_2) = \max(\nu_1, \nu_2)$	$\sigma(\cdot) \in \mathbb{R}$

Second to Second Last Step: (First) Hidden Layer to next (Second) Hidden Layer The feed-forward steps within the hidden layers are analogue to the above first step. The hidden neurons $Z_{l+1,1}, Z_{l+1,2}, ...Z_{l+1,M_l}$ are created from linear combinations of the neurons from the preceding hidden layer $Z_{l,1}, Z_{l,2}, ...Z_{l,M_{l+1}}$, such as

$$Z_{l+1,m_{l+1}} = \sigma(w_{0_{l+1,m_{l+1}}} + \sum_{i=1}^{M_l} w_{i_{l+1,m_{l+1}}} Z_{l,i}),\ m_{l+1} = 1, ..., M_{l+1} \tag{45}$$

with the same activation function as previously.
The last hidden layer passes the received and modified input variables forward to the output layer.

Last Step: Last Hidden Layer to Output Layer The target variable Y_k is modeled as a function of the derived features $Z_{L,1}, ..., Z_{L,M_L}$ of the last hidden layer.
The relations can be stated similarly to previous definition, but with o stands for output

layer:

$$Y_k = g(w_{0_{o,k}} + \sum_{i=1}^{M_L} w_{i_{o,k}} Z_{L,i}),\ k = 1, ..., K \tag{46}$$

where $g()$ is the output function for final transformation of the hidden nodes of the last hidden layer. It can be the identity function or other functions, see e.g. in [Friedman et al., 2001, p.393]).

The Learning Process of a Neural Network The mathematic description of the learning process can be found in [Friedman et al., 2001, p. 395]. The basic idea is to take the data and pass it through the neural network with some weights $w_{j_{l,m_l}}$ (starting weights can be random or chosen by the user) to get an estimated output layer. This estimated output layer is then compared to the known target values. Deviation from the known target values are errors, that can be measured e.g. with the MSE in regression or cross entropy for classification [Friedman et al., 2001, p. 395]. To minimize this error and to fit the model well to the data, the data is passed through the network again but with adjusted weights $w_{j_{l,m_l}}$. This step is repeated several times to update the weights until a stopping criterion (that can be set by the user [Friedman et al., 2001, p. 395]) is reached.

Some advantages of the Neural Network are as follows:

- Neural network are very flexibel models, they can handle misclassified training examples as well as arbitrarily complex and non-linear problems [Bischl, 2016]
- Neural network are can handle noise data
- Neural networks are a useful tool for nonlinear statistical modelling [Dau et al., 2014, p.314]
- The method doesn't need any "assumptions about the data distribution" [Dau et al., 2014, p.313]
- The method can handle high dimensional data well [Dau et al., 2014, p.314]

Neural Networks also have some disadvantages such as:

- The neural network model is generally overparametrized [Friedman et al., 2001, p.397]
- The optimization problem is nonconvex and unsTable [Friedman et al., 2001, p.397]
- The method tends to overfit the data at the global minimum of $\mathbb{R}$ [Friedman et al., 2001, p.397]
- The solution depends on the choice of starting weights [Friedman et al., 2001, p.400]

- It is difficult to analyze the network and there is no guarantees for good statistical performance [Bischl, 2016]
- For the application the input and output vectors needed to be passed as real vectors, which might be inconvenient in some examples [Bischl, 2016]

4.3.1 Neural Network for Anomaly Detection: Deep Autoencoders

A method to detect anomalies using the feed-forward neural network is called deep autoencoder [Candel et al., 2017], which is also a one-class classification method. The autoencoder consist of encoder, decoder layers, and a latent space, the latter should have a substantially lower dimension than the input layer [Goodfellow et al., 2016, p. 499]. The main difference to the classical feed-forward neural network is that the output data is set identical to the input data. The aim of training the model is to reconstruct the input (see Figure 11) with the objective to minimize the reconstruction error of the model [Dau et al., 2014, p. 314].

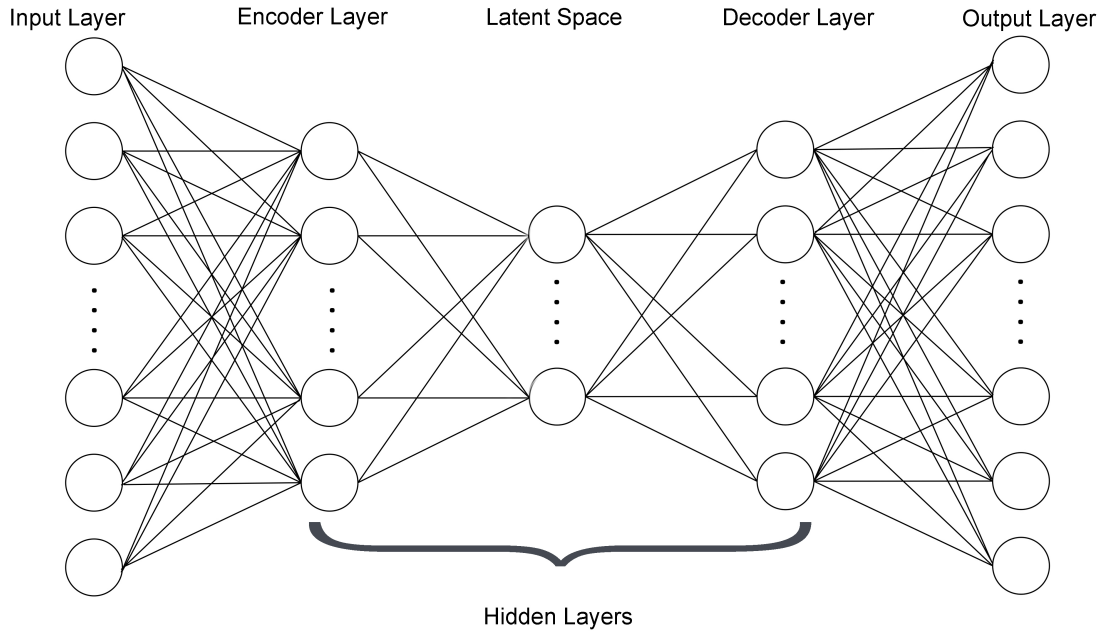


Figure 11: An exemplary autoencoder with one encoder layer, one latent space, and one decoder layer. The number of input units is equal to the number of output units.

The reconstruction error of the model is the deviation of the reconstructed input $\hat{X}$ (=output) to the true input $\hat{X}$ and is described by a loss function

$$L(X, \hat{X}) \tag{47}$$

which can be defined for example as the mean squared error (MSE). The whole process is basically to compress and decompress the data, expecting to reconstruct the normal observations well, while anomalies are poorly reconstructed [Dau et al., 2014, p. 312]. However, this approach requires to only use data of the normal class in training. The reason behind it is to make sure the network learns the pattern of the normal class. Thus the procedure classifies previously unseen observations which match the normal pattern as normal otherwise as anomalies [Dau et al., 2014, p. 311], since anomalies were not seen in training.
Furthermore, to prevent the neural network from learning the identity function, the number of neurons in at least one hidden layer needs to be of lower dimension than the input dimension. This approach forces the autoencoder to learn by non-linearly reducing the representation of the input data, which means that it learns the underlying pattern of the input data. A normal test point would match the pattern and therefore yield a rather low reconstruction error, an anomalous test point doesn't match the pattern well and therefore will likely generate a high reconstruction error [Candel et al., 2017, p. 44]. The reconstruction error of an observation should be the same as the reconstruction error of the model. For better distinction the former is called anomaly score $S = \{s_1, s_2, ..., s_N\}$ (score values for each observation in the data set). The anomaly score of an observation indicates for each instance how likely it is to be an anomaly under the learned model [Hawkins et al., 2002, p. 2]. The higher the score the more likely it is from an anomalous observation. In order to use the scores to classify the instances, the user needs to set a threshold t, which should be chosen domain specifically, i.e.

$$\begin{cases} s_i > t \text{ observation } i \text{ is an anomaly} \\ s_i \leq t \text{ normal observation .} \end{cases} \tag{48}$$

As an alternative one can set the threshold equal to the maximal reconstruction error in the training data, on the basis that it only contains normal data. Another idea is to use a specific percentile or the $mean + sd$ of the training error. Advantages of the autoencoder are the same as listed previously for neural networks in general. The disadvantages of the autoencoder are that

- there are no rules for setting the threshold for the outlier score
- it is a method for data of the semi-supervised setting, therefore it requires ideally only normal, which is not always provided data in training [Dau et al., 2014, p.4]. In real life this requirement is often not met

5 Resampling Strategies for One-Class Classification

As described in the previous sections, one-class SVM and the autoencoder are developed for the one-class classification case, which means training on data that only contains one class (the normal class), while the test set contains observations from both classes (the normal and anomaly class). The focus of this work is to assess the performance of a

learning algorithm, which is usually done with resampling strategies like cross-validation. However, the common resampling strategies like cross-validation, leave-one-out cross-validation, repeated cross-validation, out-of-bag bootstrap, subsampling or holdout (see 7.4) do not consider training only on one class but generally allowing observations of both classes in the training set. In order to perform a benchmark analysis, the above-mentioned resampling strategies need to be modified. The modified strategies are called: one-class holdout, one-class subsample, one-class bootstrap, one-class CV, one-class repeated CV.

The first three listed strategies basically sample only normal observations for the training data set and the remaining observations, including all anomalies, are used for testing. The last two strategies split the anomalous observations in k folds; every time a fold is used for training, the anomalies in this fold are dropped. `https://0xdata.atlassian.net/browse/TN-4`

6 Converting Anomaly Score Outputs into Probabilistic Outputs

Most of the current anomaly detection, algorithms return anomaly score values, such as one-class SVM, autoencoder or LOF in the previous Section 4. The scores indicate to which degree the observation is an anomaly [Gao and Tan, 2006, p.1]. For some methods, high scores are indication anomalies (e.g. LOF), for some methods low scores (e.g. OC-SVM). The scores don't have a lower nor an upper bound and vary depending on the data and the applied method (see example Figure 12), therefore it is favorable to convert these scores into probabilities [Gao and Tan, 2006, p.1].

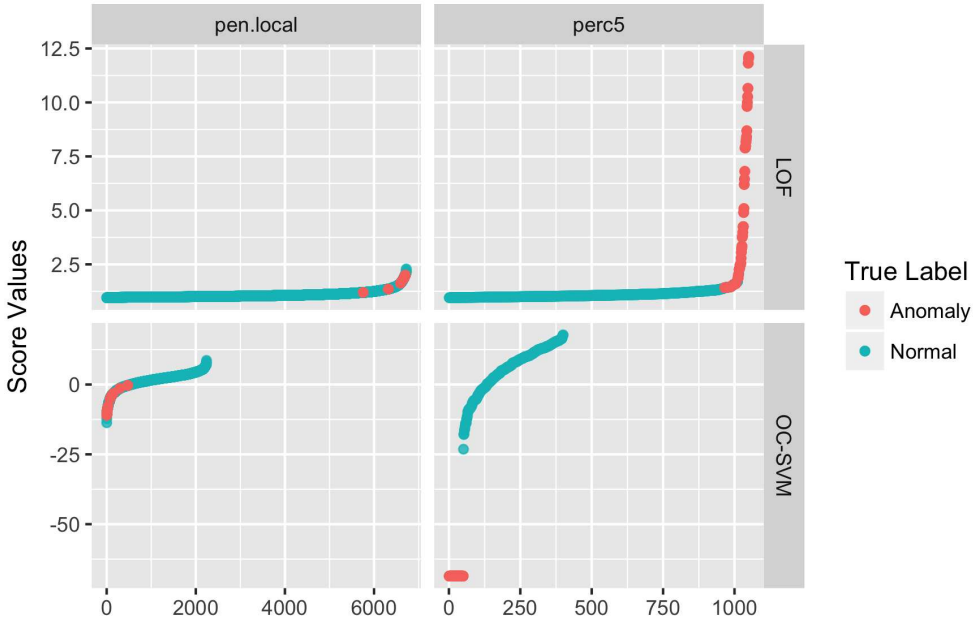


Figure 12: Sorted score values resulting from applying the OC-SVM (`kernlab` package) and the LOF (`dbscan` package) on the *perc5* and *penlocal* data sets (see Section 8.1 for data description). For LOF high scores indicate a increased likelihood of the observation beeing an anomaly, in OC-SVM low scores. The true labels of the observations are depicted by different colours.

There are advantages of having probabilities instead of scores. According to Gao and Tan [2006, p.1] those are

- It is possible to select a appropriate threshold using a Bayesian risk model.
- The interpretation of the degree of certainty is possible.
- Having the same range of $[0, 1]$ enables a comparison of methods.
- Having the same range of $[0, 1]$ it is possible to combine outputs to create an ensemble model.

Additionally, Gao and Tan [2006] show in their paper that probability scores have its advantages, for example more accurate calibration, which can be used more effectively for threshold selection and outlier detection ensemble [Gao and Tan, 2006, p.10]. Ensambles are not part of this work, however in order to use the architecture of `mlR` probability outputs are required.
The idea of converting output scores into probability scores was first introduced by Platt et al. [1999]. In their paper, Niculescu-Mizil and Caruana [2005] extend this approach to other classification methods, including neural networks.

We first introduce the basic concept of converting scores to probabilities for supervised classification. Afterwards, the unsupervised classification case is discussed.
An overview of the notation used in this chapter is given below:

- $Y = \{y_1, y_2, ..., y_n\}$ and $y_i \in \{0, 1\}$ indicating the class labels {normal, anomaly}
- $X = \{x_1, x_2, ..., x_n\}$ set of n observations drawn $\mathbb{R}^{nxd}$
- $S = \{s_1, s_2, ..., s_n\}$ anomaly scores for each observation in X. In this section high scores are considered to be an indicator for an anomalous observation.
- class-conditional densities: $p(s|y = 1)$ density of the scores conditional on the anomaly class and $p(s|y = 0)$ density of the score function conditional on the normal class
- $P(y = 1|s)$ posterior probability that x is an anomaly given the score value s
- $P(y = 0|s) = 1 - P(y = 1|s)$ posterior probability that x is a normal observation given score value s
- $\theta = (A, B)$ parameter of the logistic regression for calculating the probability distribution

6.1 Probabilistic Output for Supervised Classification

The basic idea of how to convert scores to probability in the supervised setting is to take the binary labels Y and the output scores S from a binary classification algorithm and fit a logistic regression to get the probability distribution of the classes $P(Y = 1|S)$ [Platt et al., 1999, p.2].
The requirements for applying logistic regression are

- Y is binary
- Observations $x_1, x_2, ..., x_n$ are i.i.d.

Then the predictor of the logistic regression is

$$\eta_{\theta,i} = As_i + B \tag{49}$$

with $\theta = (A, B)$. The probability distribution is then defined as a sigmoid function

$$P(Y = 1|s_i) = \frac{1}{1 + \exp(-\eta_{\theta,i})} = p_i \tag{50}$$

for which we need to estimate the parameters $\theta = (A, B)$ [Fahrmeir et al., 2007, p.189ff]. With the maximum likelihood method the parameter θ can be estimated. The underlying probability of observing y_i is a Bernoulli experiment

$$P(Y = y_i|s_i) = p_i^{y_i}(1 - p_i)^{1-y_i} \tag{51}$$

p_i depends on s_i via (50) and (49). The resulting likelihood for independently drawn observations is

$$L(\theta|Y,S) = \prod_{i=1}^{N} p_i^{y_i}(1-p_i)^{1-y_i} \tag{52}$$

and the corresponding log-likelihood-function is

$$lnL(\theta|Y,S) = \sum_{i=1}^{N} y_i ln(p_i) + (1-y_i) ln(1-p_i) \tag{53}$$

The optimal parameter θ is defined as the parameter for which the probability $P(Y|S)$ is maximized for the given observations set, where p_i depends on s_i via (50) and (49). Therefore, we maximize the Likelihood function or alternatively minimize the negative Log-likelihood

$$\hat{\theta} = \arg\max_{\theta} L(\theta|Y,S) = \arg\max_{\theta} lnL(\theta|Y,S) = \arg\min_{\theta} - L(\theta|Y,S) = \arg\min_{\theta} - lnL(\theta|Y,S) \tag{54}$$

Substituting (49),(50) into the negative Log-likelihood yields

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{N} (1-y_i)(As_i + B) + ln(1+\exp(-As_i - B)) \tag{55}$$

To solve (55) one usually sets the derivative of the likelihood function to zero. But as p_i is a nonlinear function of θ, the solution can only be found with a numeric maximization/minimization method like the Fisher-Scoring optimization or the Iterative Weighted Least Squares Algorithm [Fahrmeir et al., 2007, p.189ff].
This calibration function (called Platt Scoring) is commonly used in converting scores output from classification algorithms with known labels into probabilitie [Platt et al., 1999], [Niculescu-Mizil and Caruana, 2005].
An alternative approach to derive equation (50) through the Bayes' theorem can be found in Gao and Tan [2006]

6.2 Probabilistic Output for Unsupervised Anomaly Detection

In this work, we focus on Gao and Tan [2006] approach to convert anomaly scores into probabilities, even though different approaches are available; Gao and Tan [2006] propose two methods: method using mixture model and method using sigmoid function. The latter is implemented here and is based on Section 6.1.
One key element for converting the scores as outlined in Section 6.1 is the availability of labels. Therefore the main challenge in converting scores in the anomaly detection setting is to handle the missing of labels [Gao and Tan, 2006, p.1], which are needed in (55) to optimize the Log-likelihood. The solution proposed by Gao and Tan [2006] is to

apply the expectation-maximization algorithm, where the missing label is treated as the hidden variable.

Expectation Maximization Algorithm (EM-Algorithm) The first implementation of the EM-Algorithm was defined by Dempster et al. [1977], the following notation briefly follows his implementation but is adapted to this work.
The goal of the expectation maximization algorithm is to find the maximum likelihood estimates of parameters θ in statistical models or in other words to find the unknown probability of Y. These models usually can't be solved directly with equation (55) as described in Section 6.1, as they depend on latent variables (or hidden variables) Y, unknown parameters in θ and the observed data, which in this use case are the score values S. In equation (55) it is obvious that in order to solve the optimization problem we need the parameters θ and the latent variables Y. However, in unsupervised anomaly detection, Y is not given. Therefore, the EM algorithm can be used to split equation (53) in two interlocking equations, which means each equation needs the solution of the other equation, respectively. Following this strategy, one equation is the objective of the expectation step (E-step) and one equation will be the objective of the maximization step (M-Step). Thus each step uses the solution from the previous iteration of the respective other step. As in standard EM-Algorithm, for unsupervised case it repeatedly iterates over E and M step.

The algorithm of finding a calibration function (50), that converts scores to probabilities needs additionally following notations:

- $\hat{Y}$ is the estimation of Y, which is a probability, and not binary like Y anymore
- $\hat{\theta}$ is the estimation of θ, containing the estimated parameters for the probability distribution (50)
- l, $l \in \mathbb{N}$ indicates the iteration step

The two steps in this scenario for finding θ values for a calibration function are

- E-Step: (Updated) estimation of the latent variable Y is based on the known data S and $\hat{\theta}$ which is estimated in the previous M-step (iteration step l). The estimation $\hat{Y}$ in this step is equivalent to the temporary expectation of the probability of interest Y:

$$\hat{y}^{(l+1)} = E[y|S, \hat{\theta}^{(l)}] \tag{56}$$

- M-Step: (Updated) estimation of θ. After having an estimated $\hat{Y}$ from the preceding E-step, one can apply the ML-method as described in Section 6.1.

$$\hat{\theta}^{(l+1)} = \arg\min_{\theta} LL(\hat{\theta}^{(l)}|\hat{y}^{(l+1)}, S) \tag{57}$$

[Gao and Tan, 2006, 3]
Some additional comments on the algorithm stated in [Gao and Tan, 2006]:

- In the first iteration of the EM loop we need to initialize the parameter $\theta^{(0)}$. The value can be chosen based on experience or randomly [Aggarwal, 2017]. Within this work we choose the initial parameter as $\theta^{(0)} = (A, B) = (1, 0)$, which is a common choice.
- Iterate the EM loop until convergence, i.e. until the objective function in the M-Step ceases to change, which means $\theta^{(l+1)} - \theta^{(l)} \leq \epsilon$ with ϵ some tolerance parameter.
- It is proven that the EM-Algorithm always converges, but it only finds local optima, which doesn't change the optimal solution in this case as the likelihood function is concave and therefore only has one unique solution.
- The calibration function is monotonously increasing, thus it doesn't change the ordering of the original score values. In mlR however, the score outputs of some methods (e.g. `ksvm` in the `kernlab` package) are reversed to yield an uniform interpretation, i.e., high probabilities indicate that the observation is likely to be an anomaly.

Note: Unlike in theory, in practice the models do not converge, when applying this method. In `mlR` we therefore fix the parameters to $(A, B) = (1, 0)$. Figure 13 displays the converted score values from Figure 12.

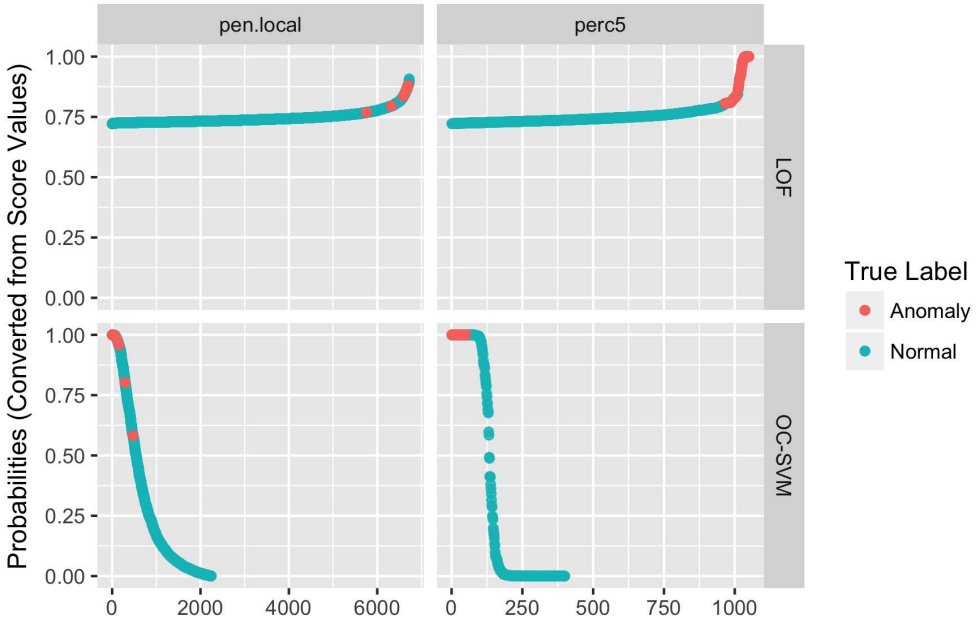


Figure 13: Sorted score values converted to probabilities, resulting from applying the OC-SVM (`kernlab` package) and the LOF (`dbscan` package) on the *perc5* and *penlocal* data sets (see Section 8.1 for data description). The parameter of (50) is set to $(A, B) = (1, 0)$. For LOF, high probabilities scores indicate that the observation is rather an anomaly, in OC-SVM low scores probabilities. The true labels of the observations are depicted by different colours.

Note: The interpretation of the probabilities converted from scores is that high probabilities indicate that the observation is anomalous, therefore the original score values need to have the same direction of the interpretation. However, if the score values have an inversed direction of interpretation, the resulting probability after converting the scores will be inversed (e.g. when implementing OC.SVM in mlR).

7 Open Source `R` Package: `mlR`

7.1 What is `mlR`

`mlR` is short for Machine Learning in `R`. It provides an interface to a large number of machine learning techniques in `R`. The main goal is to provide a unified interface for machine learning tasks [Bischl et al., 2016]. Implemented are supervised methods such as regression and classification tasks and unsupervised methods such as cluster analysis, but also survival analysis and general, example-specific cost-sensitive learning.
When using machine learning methods to solve a task, it is mostly not obvious which method and method setting is suitable to do so. Therefore, it is often necessary to carry

out standard methods like resampling and hyperparameter tuning to find the optimal learner and parameter setting of the learner. But there is a lack of common interfaces to execute the work stages conveniently. [Bischl et al., 2016] introduce `mlR` to solves this problem.
A benefit in mlR is, that it provides a uniform framework to handle machine learning tasks, it basically offers a framework which can be connected with already established packages in `R`. `mlR` covers techniques from generic resampling to cross-validation, bootstrapping, subsampling, tuning, feature selection, and model comparison. All implemented in generic building blocks, which simplifies the use of machine learning methods and makes their use more intuitive. Additionally, `mlR` offers easy parallelization for most of the methods. Due to the general structure of the package, more advanced users are enabled to extend available algorithms or embed their own methods, while still using the `mlR` infrastructure. The user can also connect the OpenML `R` Package [Bischl et al., 2016] to use open source data.
A list of features offers in `mlR`:

- Possibility to fit, predict, evaluate and resample data
- Easy extension mechanism through S3 inheritance
- Abstract description of learners and tasks by properties
- Parameter system for learners to encode data types and constraints
- Many convenience methods and generic building blocks for machine learning experiments
- Resampling like bootstrapping, cross-validation and subsampling
- Different visualizations of predictions ,e.g., ROC curves
- Benchmarking of learners for multiple data sets
- Easy hyperparameter tuning using different optimization strategies
- Variable selection with filters and wrappers
- Nested resampling of models with tuning and feature selection
- Cost-sensitive learning, threshold tuning and imbalance correction
- Wrapper mechanism to extend learner functionality and complex and custom ways
- Combine different processing steps to a complex data mining chain that can be jointly optimized
- Extension points to integrate own methods
- Parallelization is built-in

The `mlR` tutorial website ([mlr]) provides detailed examples and explanation on how to use those features.

7.2 General Structure of `mlR`

First, we want to provide an overview of the very basic structure of how to conduct a machine learning analysis with `mlR`. Later options are described to extend the analysis e.g. tuning of the hyperparameters. To show that the generic building blocks of `mlR` is quite intuitive to apply, we will explain the application with a question-answer style.

1. Define the task:
   - What is the problem/task you want to solve?
     (`Classif` =classification, `Regr` =regression, `Cluster` clustering, `CostSens` = cost sensitive learning, `Multilabel` = multilabel classification, `Surv` = survival learning, `OneClass` = one-class)
   - What data do you use?
   - Do you have a target variable (supervised) or not (unsupervised)? If former, what is the name of your target variable?

```
# load package and data
library(mlr)
data(iris)

# make<Task type>Task(id = <set an ID name>,
# data = <used data set>,
# target = <target variable name>), e.g.
task = makeClassifTask(id = "tutorial", data = iris,
  target = "Species")
```

2. Define the learner:
   - What learner do you want to use? (random forests, boosting, SVM, etc.)
   - Assume you want to model a SVM. Do you already have a prefered package in `R`, which can model a SVMs? e.g. `e1071`? Check `http://mlr-org.github.io/mlr-tutorial/devel/html/integrated_learners/index.html` for the learners name in `mlR`.
     If you don't have a preferred package, check what learners are available at the same URL.

```
# makeLearner("<learner name>")
lrn = makeLearner("classif.svm")
```

3. Create train and test data:
   - If you have pre-settings about which instances are in the test or train data, assign the indices accordingly, otherwise sample the indices randomly.

```
n = nrow(iris)
train.set = 1:100
test.set = 101:150
```

4. Fit the model:
   - Train the learner on the task and the specified training data

   ```
   model = train(lrn, task, subset = train.set)
   ```

5. Make predictions:
   - After training the model, you want to know what the predicted values are for new observations, therefore pass indices of the test set.

   ```
   pred = predict(lrn, model, subset = train.set)
   ```

6. Evaluate the learner:
   - You can choose a performance measure to evaluate how well your learner performed. If you didn't decide on the performance measurement yet, you can use the default performance measure, which is in this case the mean classification error (mmce),

   ```
   getDefaultMeasure(task)
   # mmce
   ```

   alternatively `mlR` can show you suitable possibilities for your task with

   ```
   listMeasures("classif")
   ```

   If you can't decide, compare more than one performance measure, e.g. mean classification error (`mmce`) and accuracy (`acc`)

   ```
   performance(pred, measures = list(mmce, acc))
   ```

For more examples of the basic structure visit `http://mlr-org.github.io/mlr-tutorial/release/html/` and for more details about possible modifications of the used functions read the `mlR` help pages of each function. This example is only a small part of what `mlR` is capable of.

7.3 Anomaly Detection in `mlR`

Within the scope of this work methods for anomaly detection were embedded into the `mlR` package. The methods include support vector machines for one-class classification (anomaly detection) from the `e1071` package [Meyer et al., 2015] and from the `kernlab` package [Karatzoglou et al., 2004], an autoencoder from the `h2o.deeplearning` package [Candel et al., 2017], the local outlier factor method from the `DMwR` package [Torgo, 2010] as well as from the `dbscan` package [Hahsler and Piekenbrock, 2017]. If you prefer other

techniques for anomaly detection, which are already implemented in R but not in mlR, you can follow the instructions on http://mlr-org.github.io/mlr-tutorial/devel/html/create_learner/index.html to implement your own learner.

7.3.1 New and modified Functions for Anomaly Detection in mlR

In order to make anomaly detection work in mlR a series of R files in the mlR repository needed to be modified or added. There are R files, that contain new and modified functions (Table 27) and test-files to test these (Table 29). For the testing a new synthethic data set *oneclass2d.task.Rdata*, with two features, 1000 normal an 50 anomalous observation is created (see ?oneclass2d.task.Rdata in R for more details).
The main functions added to mlR are the unsupervised anomaly detection learners:

- makeRLearner.oneclass.svm() based on the e1071 package
- makeRLearner.oneclass.ksvm() based on the kernlab package
- makeRLearner.oneclass.lofactor() based on the DMwR package
- makeRLearner.oneclass.lof() based on the dbscan package
- makeRLearner.oneclass.h2o.autoencoder() based on the h2o package

These learners can be applied to one-class tasks, created with the new makeOneClassTask()-function. Although the training is unsupervised, the task is implemented with the option to add a target column for supervised evaluation of the test set (if labels are available). One-class Resampling strategies (more on one-class resampling in Section 5) can be used to assess the performance of anomaly detection methods, e.g. with following measures:

- makeAMVMeasure to create AUMVC (Section 3.2.2)
- makeAMVhdMeasure and makeAMVhdWrapper to create AUMVChd (Section 3.2.2.3)
- makeWACMeasure to create WAC (Section 3.1)
- makePrecisionMeasure to create different precision measures (Section 3.1)

The respective help pages provide more description and details on application. A more compact and quick explanation on how to apply mlR for anomaly detection is provided in Section 7.4.
Every added learner is based on an already well-established R package. The following subsection gives a short description of those packages.

7.3.2 Package `e1071`

The package `e1071` provides an interface in `R` to C++ libary libsvm [Meyer et al., 2015, vignettes p.2]. Besides of classification (C- and ν-classification) and regression (ϵ- and ν-regression), it also covers anomaly detection (one-class-classification). As described in chapter 4.1, SVMs use kernels to expand to nonlinear class boundaries according to Schölkopf et al. [2000], Chang and Lin [2011]. In `e1071` the linear, polynomial, radial basis function and sigmoidal kernels are included and their formula are described in Table 2.
For more details see the help page of Meyer et al. [2015, vignettes p.7]. A drawback of SVM in `e1071` is that the current implementation is only optimized for the radial basis kernel function [Meyer et al., 2015, vignettes p.8].

7.3.3 Package `kernlab`

The package `kernlab` contains kernel-based machine learning methods for `R`. It provides an implementation of the SVM method. It enables the user to extend the code e.g. add new kernels or different optimizers [Karatzoglou et al., 2004, p.2]. `kernlab` already includes some basic kernel functions implementation see Table 2 and additionally Table 4

Table 4: Addtional kernel functions available in package `kernlab` [Karatzoglou et al., 2004]

Kernel	Formula	Parameters
Bessel	$\frac{Bessel^n_{\nu+1}(\sigma\|\|x-x'\|\|)}{(\|\|x-x'\|\|)^{-n(\nu+1)}}$	σ, ν
Laplace Radial Basis Fct.	$exp\{-\sigma\|\boldsymbol{u}-\boldsymbol{v}\|\}$	σ
ANOVA radial basis fct.	$\left(\sum_{k=1}^{n} exp\{-\sigma\|\boldsymbol{u}_k-\boldsymbol{v}^k\|_2\}\right)^d$	σ, d

7.3.4 Package `DMwR`

The package `DMwR` is one of the first packages in `R` containing LOF, it is used in Zhao [2012] and in Torgo [2010] to demonstrate data mining in R, including outlier detection. Besides outlier detection, it also contains functions used in the book Torgo [2010], such as kNN, SVMs, random forests and more.

7.3.5 Package `dbscan`

The package `dbscan` provides a fast `C++` implementation of several density-based algorithms, such as the DBSACN, OPTICS, HDBSCAN as well as the LOF method. According to Hahsler and Piekenbrock [2017] faster than native `R` implementations as well as `WEKA` ([Frank et al., 2009]), `ELKI` ([Achtert et al., 2008]) and Python's scikit-learn

([Pedregosa et al., 2011]). When calculating the LOF of each data point, `dbscan` uses a kd-tree (space-partitioning data structure) for faster k-nearest neighbor search instead of a linear search [Hahsler et al., p.10], that approach avoids to calculate the complete distance matrix, and therefore save computational time.

7.3.6 Package `h2o.deeplearning`

`H2o` is a machine learning platform, that can be used with the web browser, but also with Python, Apache Hadoop, Spark and `R` with the `R` package `H2o`. It is convenient as analyzing the data sets are held in cloud computing systems [Candel et al., 2015]. Currently different algorithms are included, ranging from classical statistical analysis methods (e.g. GLM) to Clustering (k-means), ensembles (e.g. random forest) and deep neural networks, including the autoencoder [h2o], which is implemented as a one-class learner within this work.

7.4 Application of Anomaly Detection in `mlR`

An extensive explanation of how to apply anomaly detection methods from `mlR` can be found on the `mlR` tutorial website [mlr] or under this link `https://github.com/mlr-org/mlr-tutorial/blob/gh-pages_oneclass/src/oneclass_classification.Rmd`.

8 Benchmark Setup

8.1 Data Sets

There is no typical, established benchmark data set for anomaly detection yet, and there is also a lack of open-source data available for anomaly detection. Therefore, we apply our methods to several data sets used in other papers as well as on simulated anomaly data sets (see simulation structure below). All data sets used were modified to have a target column called "Target" containing class labels "Anomaly" and "Normal". Those data sets will be publicly available at `openml.org` [Vanschoren et al., 2013]. An overview of the data is provided in Table 5.

Name of Data Set	Number of Observation	Number of Features	Share of Anomalies	Synthetic Data
banana	5300	2	45%	no
artificial.unsup	3000	2	1.2%	yes
perc5	1050	4	4.7%	yes
perc10	1100	4	9%	yes
annthyroid	7299	6	8%	no
mammography	11183	6	2.3%	no
perc5hd	1050	10	4.7%	yes
perc10hd	1100	10	9%	yes
penlocal	6724	16	0.15%	no
penglobal	809	16	11%	no
letter	1600	32	6.25%	no
satellite	5100	36	1.5%	no

Table 5: Overview of data sets (synthethic and open source) used in the benchmark experiments within this work. All data sets were modified to have a target column called "Target" containing class labels "Anomaly" and "Normal".

8.1.1 Synthetic Data Sets

perc5(.hd), perc10(.hd) (low/high dimensional) To create synthetic anomaly data, 1000 uncorrelated observations were drawn from a multivariate normal distribution and sample variance between 1 and 9, and 4 (*perc5*, *perc10*) or 10 feature variables (*perc5.hd*, *perc10.hd*). Data with ten features will be considered as high dimensional data. Samples of 5% or 10% anomalous observations were drawn randomly of discrete numbers between 2 and 100, with replacement.

artificial.unsup The *artificial.unsup* data set is based on four normal distributions (one of which with low density), a micro cluster and local anomalies. Currently, the data set is available from the authors, but it is available at `http://www.madm.eu/downloads`.

8.1.2 Open Source Data Sets

banana The *banana* data set is used in [Thomas et al., 2016]. It will be available on openML [Vanschoren et al., 2013].

annthyroid The *annthyroid* data set is a modified version of the "Thyroid Disease" data set from the UCI machine learning repository. It is available at `http://archive.ics.uci.edu/ml/datasets/mammographic+mass`.

mammography The *mammography* data set is used in [Root et al., 2015] and is a modified version of the openly available data set in the UCI machine learning repository. The modified version is available at `https://www.cs.sfu.ca/~gza11/personal/research/mass/`.

penglobal The *pen.global* data set is a modified version of the "Pen-Based Recognition of Handwritten Digits" data set, available in UCI machine learning repository and is used in [Goldstein and Uchida, 2016, p.18]. It is available at `http://www.madm.eu/downloads`.

penlocal The *pen.local* data set is a modified version of the "Pen-Based Recognition of Handwritten Digits" data set, available in UCI machine learning repository and is used in [Goldstein and Uchida, 2016, p.18]. It is available at `http://www.madm.eu/downloads`.

letter The *letter* data set is used in [Goldstein and Uchida, 2016, p.18]. It originally contains 16 features from the English alphabet and was modified for unsupervised anomaly detection. It is available at `https://dataverse.harvard.edu/file.xhtml?fileId=2711926&version=RELEASED&version=.0`

satellite The *satellite* data set is a modified version of the "Statlog (Landsat Satellite)" data set, available in UCI machine learning repository. It is available at `http://www.madm.eu/downloads`.

For further and more detailed information, the reader should follow the links and references provided for the respective data sets.

8.2 Method and AUMVC(hd) Settings

The frameworks used in the experiments are provided in Table 6. We compare the OC-SVM (Section 4.1) to the LOF method (Section 4.2) from the `kernlab` package and the `dbscan` package, respectively. The OC-SVM is designed for training on one class, therefore we use OC-CV as resampling method. However, in real life it is not always guaranteed that we only have normal observations in the training data. To test the OC-SVM method under real life circumstances we also execute the experiment with CV. The LOF method is an unsupervised anomaly detection method with no restriction on the training data, hence only resampling method CV is used.
Every experiment will be tuned and evaluated with both the AUC and the AUMVC(hd) (Section 3.2). We also compare them to experiments where we don't tune the hyper parameters at all. If we tune and evaluate with the AUC, we will assume to have labels for testing, if we evaluate with AUMVC we don't use labels at all. The results from not tuning are the lower bound, the results from AUC the upper bound, both are benchmark values to evaluate whether AUMVC is a useful tuning measurement. Although it should be noted that in some examples tuning with AUC (and labels) is outperformed by no tuning or tuning with AUMVC(hd).

Methods	Resampling Strategy (Inner and Outer)	Tuning Search Space		Search Type	Rest Parameter	Tuning and Evaluation Measurement	Data Set
OC-SVM	3-fold CV and 3-fold OC-CV	ν	Lower Bound = 0.01 Upper Bound = 0.99	Random Search	Default of ksvm() from the `kernlab` R package	No Tuning, AUC, AUMVC(hd)	All 12 Data Sets
		σ	Lower Bound = -12 Upper Bound = +12				
		with Transformation Function for σ: $f(\sigma) = \sigma^2$					
LOF	3-fold CV	k	Lower Bound = 10 Upper Bound = 150		No other Parameter		

Table 6: Setting of the benchmark experiments.

We choose LOF from the `dbscan` package, as the implementation based on `C++` is much faster than in the `DMwR` package. The OC-SVM in `e1071` and `kernlab` are similar in terms of computational time [Karatzoglou et al., 2005], the `kernlab` package is chosen by personal preference.
The default setup of the new introduced measurement AUMVC and AUMVChd needs to be determined too. After conducting performance and runtime analysis in Section 9.1, the measures' settings are set to the final values (in times of our conducted analysis) in Table 7 for the final evaluation. The α interval was set to $[0.9, 0.99]$ due to recommendation in [Goix, 2016, p.2].

Measurement	Parameter	Description (see Section 3.2.2.4)
AUMVC and AUMCVhd	alpha = $[0.9, 0.99]$	The interval to take the integral of the MV curve $[\alpha_1, \alpha_2]$
	n.alpha = 50	Number of splits of the alpha interval n_α
	n.sim = 10^3	Number of Monte-Carlo samples n_{sim}
AUMVChd	amv.iters = 20	Number of internal subsamples m
	amv.feats = 5	Number of feature sub-sampling d'

Table 7: Settings of the tuning and evaluation of measurements AUMVC and AUMVChd. The AUMVChd has two additional parameters.

9 Benchmark Results

As described in Table 6, there are many different experiment settings. In order to keep the description of the results as clear as possible, the abbreviations used in this section are introduced in the following table.

Abbreviation	Description
OC-SVM_{cv}	OC-SVM with 3-fold CV in the outer and inner resampling loop
OC-SVM_{occv}	OC-SVM with 3-fold OCCV in the outer and inner resampling loop
LOF_{cv}	LOF with 3-fold CV in the outer and inner resampling loop
AE_{cv}	Autoencoder with 3-fold CV in the outer and inner resampling loop
AE_{occv}	Autoencoder with 3-fold OCCV in the outer and inner resampling loop
R_{cv}	Resampling with 3-fold CV
R_{occv}	Resampling with 3-fold OCCV
T_0	No tuning
T_{AUC}	Tuning with AUC
T_{AUMVC}	Tuning with AUMVC
$T_{AUMVChd}$	Tuning with AUMVChd

Table 8: Abbreviations for different settings.

9.1 Performance and Runtime Analysis of AUMVC(hd)

In Section 3.2, the theory of $AUMVC$ and $AUMVC$(hd) was introduced, one of the very few performance measurements for unsupervised methods. The aim of this benchmark analysis is to find the optimal parameter settings for those measurements and to compare the $AUMVC$(hd) to the supervised measurement AUC in terms of runtime and performance depending on different parameter settings.

9.1.1 Runtime Analysis

The two objectives of the following runtime analysis are to understand

1. How computationally expensive is the AUMVC(hd) in comparison to the AUC?
2. How do the parameters of AUMVC(hd) and the size of the data set impact the runtime of the measurements?

For the analysis, synthetic data sets were simulated according to Section 8.1.1 with 5% anomalies, and different data set sizes, the true column is additionally saved for evaluation. The predicted response was calculated by using a OC-SVM with default settings. The experiments are only based on the calculation of the performance measurement, excluding the runtime for executing the anomaly method, since we are only interested in the computational cost of the measurement itself in this section.
The tested parameters of the AUMVC(hd) measurement are listed in Table 9.
The analysis was run twice. For the first iteration a higher number of different parameter settings were tested. In order to keep the tables and figures simple, parameters that appeared to not have a noteworthy impact on the runtime were reduced to less examples.

Varaible	Value	Default	Measure
Number of Monte-Carlo Samples (n.sim)	$\{10, 10^2, 10^3, 10^4\}$	default:10^3	AUMVC/AUMVChd
Number of splits of the alpha interval (n.alpha)	$\{10, 20, 30, 40, 50\}$	default: 50	AUMVC/AUMVChd
Number of observations in data n (n)	$\{1050, 10500\}$	default: 1050	AUC/AUMVC/AUMVChd
Number of features in data d for AUMVC (d)	$\{2, 4, 6\}$	default: 2	AUC/AUMVC
Number of features in data d for AUMVChd (d)	$\{8, 15, 20\}$	default: 8	AUC/AUMVChd
Number of internal features in subsamples in AUMVChd (amv.feats)	$\{2, 5\}$	default: 5	AUMVChd
Number of internal subsamples in AUMVChd (amv.iters)	$\{5, 20, 50\}$	default: 50	AUMVChd

Table 9: Overview of settings for the runtime analysis of AUMVC and AUMVChd.

The violin plot in Figure 14 shows the runtime results (in seconds) of the AUMVC performance in comparison to AUC as well as different settings of AUMVC. Table 11 summarises the numerical effects (Appendix Table 30) on runtime and is displayed in Figure 14. Each experiment is evaluated 100 times and visualsed as one violin, representing the distribution (in seconds) of the runtime of the experiments.
The names on the y-axes indicate the deviation from the default setting, examples on how to read the axis are stated in Table 10.

The experiment was based on a synthetic data set with 6 features, and 1050 observations, evaluated with default AUMVC. All other settings are according to the default values in Table 9

Name of y-axis	Description
dim6.n1050.aumvc	The experiment was based on a synthetic data set with 6 features, and 1050 observations, evaluated with default AUMVC. All other settings are according to the default values in Table 9
alpha10.aumvc	The experiment was executed with default AUMVC, but with $n.alpha = 10$. All other settings are according to default value in Table 9.

Table 10: Example of how to read the y-axis from Figure 14 and Figure 15. The labels on the y-axis indicate the deviation from the default setting in Table 9.

The default AUMVC (*dim2.1050.aumvc*) is ~30 times slower than AUC (for data size 1050), which is expected as AUMVC uses Monte-Carlo integration. When increasing the data size to 10500 observations, the runtime of using AUMVC (*dim2.10500*) is ~112 times slower than with AUC. Increasing the number of Monte-Carlo samples from 10 to 100 (from 100 to 100000) increases the runtime by factor ~1.2 (~9.7). The runtime effect seems to increase over-proportional by increasing the number of Monte-Carlo samples. However, the number of alpha splits seems to have no major affect on runtime. The number of features in the data also has little effect. By contrast, increasing the number of observation by a factor of 10 leads to a ~6 times slower runtime.
In summary, the runtime bottleneck of AUMVC is the number of Monte-Carlo samples.

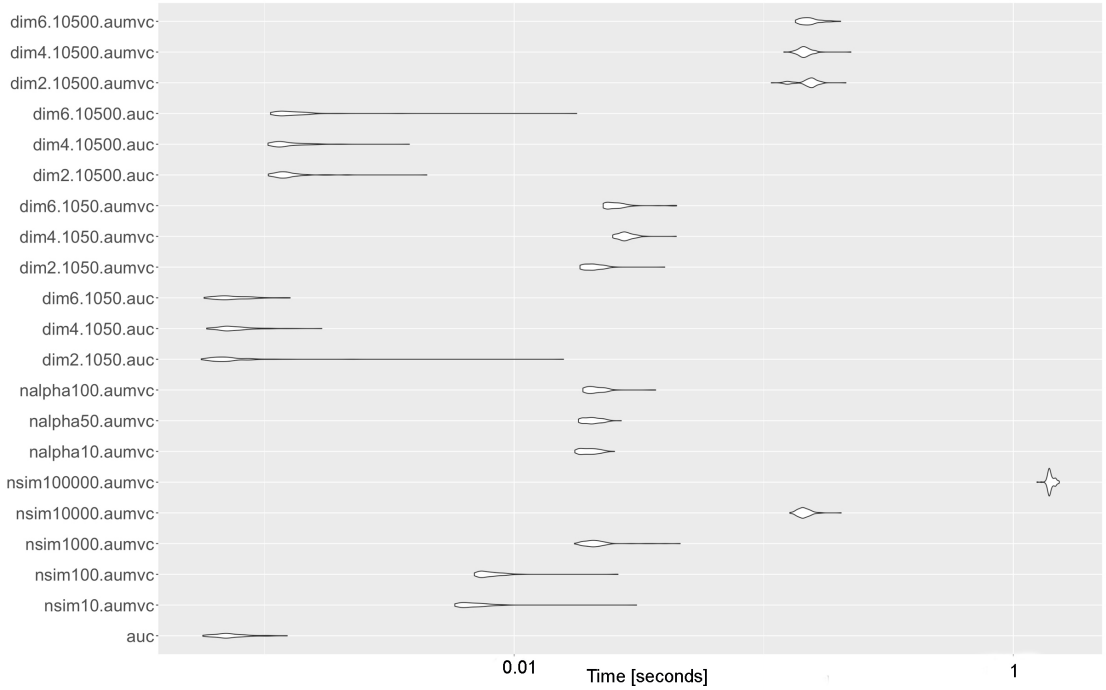


Figure 14: Comparison of runtime (in seconds) for different settings for AUC and AUMVC, each experiment was repeated 100 times. The labels on the y-axes state the values of the parameters that is the focus of the experiment or deviates from the default setting (one or more parameter possible) (see examples how to read the y-axis in Table 10). The corresponding numeric values are in Appendix 30.

Tested Data Size / and Parameter of AUMVC	Factor of Increasing the Tested Value	Experiment Comparison (y-lab names)	Factor of Mean Effect on Performance AUMVC (AUC)	Direction of Effect
n	10	*dim2.10500.aumvc* to *dim2.1050.aumvc* (analog for *dim4.* and *dim6.*, and for *.auc*)	~6 (~1.5)	increasing
d	2 (3)	*dim2.1050.aumvc* to *dim4.1050.aumvc* and to *dim6.1050.aumvc* and also for n = 10500)	~1	steady
n.sim	10 to 100 (10000 to 100000)	*nsim10.aumvc* to *nsim100.aumvc* (analog for n = 10500)	~1.2 (~9.7)	increasing
n.alpha	5 (10)	*nalpha10.aumvc* to *nalpha50.aumvc* and to *nalpha100.aumvc*	~1	steady
AUC to AUMVC n = 1050 (n = 10500)		*dim2.1050.aumvc* to *dim2.1050.auc* (analog for *dim4* and *dim6*, and for n=10500)	~30 (~112)	increasing

Table 11: Compact summary of the numerical effects in Figure 14.

An equivalent runtime analysis is also executed for AUMVChd on high-dimensional data (see results in Figure 15, Table 12 and Table 31). Since the number of alpha splits doesn't have an affect on the runtime of AUMVChd (similar to the runtime of AUMVC as the former is a repetition of the latter), it is excluded from Figure 15. The number of features in the data also doesn't have an affect on the runtime of AUMVChd, as internally the AUMVC is evaluated on a subsample with fewer features (d' = *amv.feats*), which is the reason why it is also excluded in Figure 15 for a better overview.
The results show, that the default AUMVChd (*n.1050.aumvc* and *n.10500.aumvc*) is ~4497 times slower than AUC (*n.1050.auc*) in the smaller data set and ~122021 times slower than AUC in the bigger data set. Increasing the number of Monte-Carlo samples from 10 up to 100 seems to have nearly no effect, but increasing the number from 1000 to 10000 (from 10000 to 100000) leads to a higher (over-proportional) effect on runtime by factor ~3 (~9.7). Increasing the number of internal subsamples of AUMVChd increases the time, but the effect is not increasing linearly, as increasing *amv.iters* by 5 increases the time by ~1.9 and increasing *amv.iters* by 10 increases the time by ~3.3. The effect on runtime when increasing the number of observations in the data set by factor 10 is ~40, which is a stronger gradient than when using AUMVC.
In summary, the bottleneck of the runtime of AUMVChd is the size of the data and additionally the number of Monte-Carlo samples (same as for AUMVC) and the number of internal subsamples.

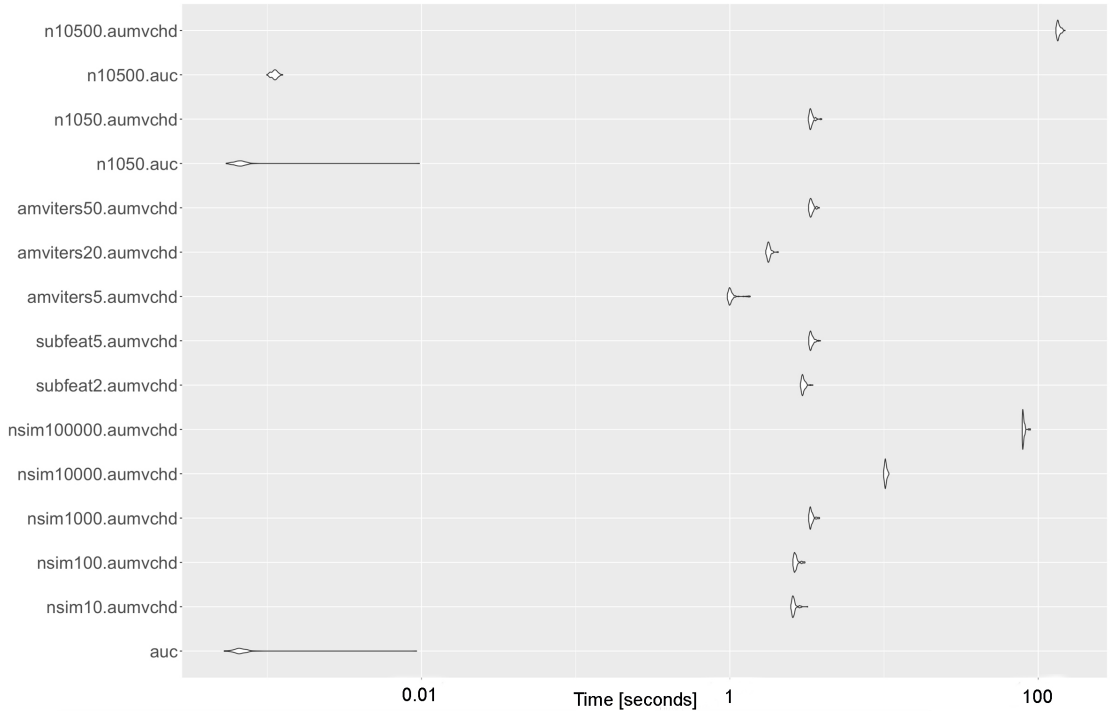


Figure 15: Comparison of runtime (in seconds) for different settings for AUC and AUMVChd, each experiment was repeated 100 times. The labels on the y-axes state the values of the parameters that deviates from the default setting (see examples how to read the y-axis in Table 10). The corresponding numeric values are in Appendix 31.

Tested Data Size / and Parameter of AUMVC	Factor of Increasing the Tested Value	Experiment Comparison (y-lab names)	Factor of Mean Effect on Performance AUMVChd (AUC)	Direction of Effect
n	10	*n10500.aumvchd* to *n1050.aumvchd* (analogue and for *.auc*)	~40 (~1.5)	increasing
n.sim	10 to 100 (10000 to 100000)	*nsim10.aumvchd* to *nsim100.aumvchd* (analogue for n = 10500)	~1.2 (~9.7)	increasing
amv.feats	2.5	*subfeat2.aumvchd* to *subfeat5.aumvchd*	~1.1	increasing
amv.iters	5 (10)	*amviters5.aumvchd* to *amviters20.aumvchd* and to *amviters50.aumvchd*	~1.7 (~3.3)	increasing
AUC to AUMVChd n = 1050 (n = 10500)		*n1050.auc* to *n1050.aumvc* (analogue for n=10500)	~4497 (~122021)	increasing

Table 12: Summary of the numerical effects in Figure 15.

In both evaluation methods AUMVC and AUMVChd the number of Monte-Carlo samples has a strong effect in comparison to other settings, therefore the performance of this parameter for AUMVC will be further investigated in Section 9.1.3. The parameters *n.alpha* and *amv.iters* were also investigated due to theory based reasons (see Section

9.1.4 and 9.1.5).

9.1.2 Fluctuation of the AUMVC(hd)

As the AUMVC and AUMVC(hd) sample observations from the hypercube of the data to calculate the performance values, the measures value might fluctuate depending on the subsamples. For the analysis of this behaviour we only look at the performance evaluation of a predicted response. In order to achieve this, we take the same synthetic data as in the previous Section 9.1.1 with 6 and 20 dimension to evaluate 100 times the default AUMVC and default AUMVChd, respectively.
As a reminder: Default AUMVChd evaluates the default AUMVC on 50 feature subsamples, each with 5 features. The default AUMVC uses 1000 Monte-Carlo samples to estimate the area under the mass volume curve. The AUC does not depend on a random component, thus its standard deviation is 0.
Figure 16 shows that the AUMVC and AUMChd values do fluctuate and the interquartile range of AUMVC is similar, but a little bit larger than that of AUMVChd. The effect of AUMVChd's lower fluctuation might occur due to the 50 internal repetitions of the AUMVC measure within AUMVChd. In other words, AUMVChd is already an average value of AUMVC, therefore AUMVChd might be more stable. On the other hand AUMChd has one more random component, namely the drawing of subfeatures. It is recommended to made more investigation to yield a clear statement, which is not done within this work.
Another analysis regarding the fluctuation of AUMVC is shown in Figure 17, that is based on the evaluation of AUMVC and AUC using a R_{cv} for six low dimensional data sets (described in Section 8.1). The y-axis value shows the standard deviation of the AUMVC and AUC over 3 resamples. It should be noticed that the scale of AUMVC differs from that of AUC and that AUMVCs standard deviation (in this benchmark experiment) lies between 0 and 2.7e+05, while the AUCs lies between 0 and 0.1. However, the AUC has an upper bound of 1, while AUMVC doesn't have an upper bound.
In conclusion the only fact is, that AUMVC(hd) does fluctuate even when evaluated on the same data, which doesn't have to be a disadvantage if using AUMVC(hd) for tuning improves the performance of used methods (see analysis in Section 9.2), as that's the main purpose of this measure.

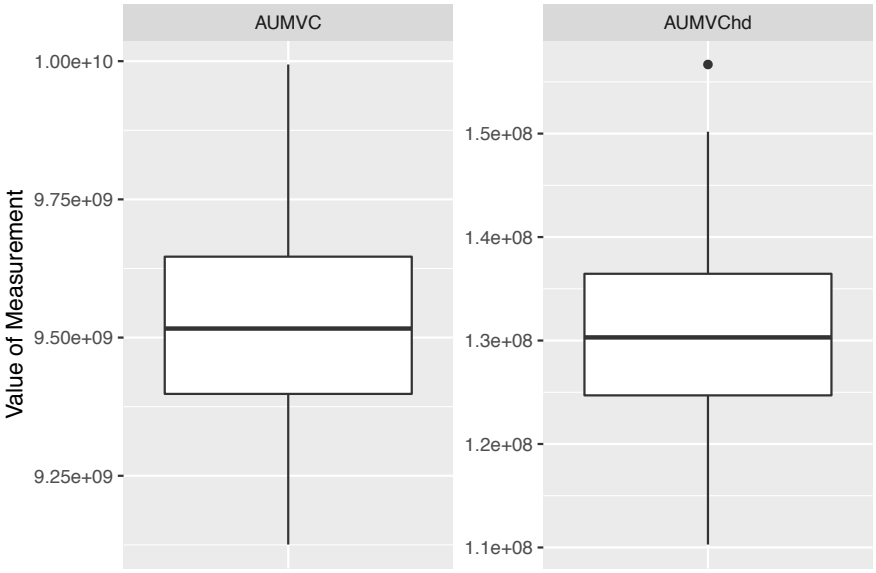


Figure 16: Boxplot displaying the fluctuation of the AUMVC value and AUMVChd value, when evaluating the performance on the same predicted response 100 times. The data set for evaluating AUMVC (AUMVC(hd)) is a synthetic data set with 6 (10) features, 1050 observations and 5% anomalies. The predicted response was calculated by a OC-SVM with default setting.

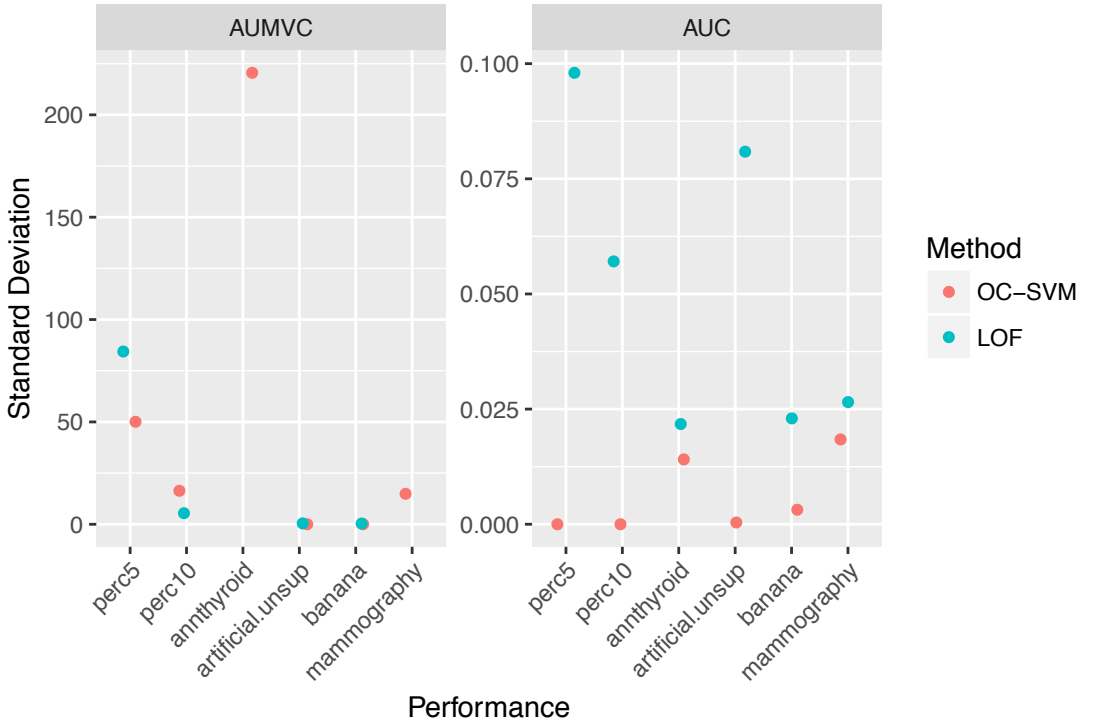


Figure 17: Standard deviation of the AUMVC and AUC measure over three folds of R_{cv} for 6 different anomaly data, grouped by methods (OC-SVM_{cv} and LOF_{cv}). Two standard deviation values were deleted to make the figure visible (standard deviation of LOF_{cv} on *annthyroid* data (2.718459e+05) and on *mammography* data (3.340673e+04))

9.1.3 AUMVC(hd) Performance w.r.t. Number of Monte-Carlo Samples

This subsection investigates the performance of AUMVC in dependence of the number of Monte-Carlo samples since the previous section identified that parameter as the bottleneck of the runtime.
AUMVC lies in $[0, \infty]$, where 0 is the best case. This analysis includes calculations with OC-SVM_{cv} and LOF_{cv} on three different data sets: *perc5*, *mammography*, and the *annthyroid* data set (see Section 8.1 for details). Each combination of data set and method was repeated 6 times, each time with another tuning measurement: T_0, T_{AUC} as well as T_{AUMVC} with 10, 100, 1000, 10000 number of Monte-Carlo samples. Additionally, all measurement were evaluated for each experiment. As the AUMVC value depends on random draws, each experiment was repeated 3 times with different seeds (see settings in Table 13).

Data Set	*perc5*, *mammography*, *annthyroid*
Methods	OC- SVM, LOF
Resampling	CV
Tuning Measurement	T_0, T_{AUC}, T_{AUMVC} with Monte-Carlo samples $n.sim \in \{10, 100, 1000, 10000, 10000\}$
Number of Experiment Repetition	3 times with different seeds

Table 13: Overview of the experiment settings for analysing the effect of the Monte-Carlo sample size in AUMVC(hd).

The default setting for OC-SVM is a Gaussian kernel with $\nu = 0.2$ and $\sigma = 0$ and the default setting for LOF is that the variable k (indicating the number of neighbors) is set to 20.
Table 32 and Table 33 in the Appendix show the results for applying OC-SVM_{cv} and LOF_{cv} respectively for every repetition, whereas Table 14 and Table 15 display the mean of the repetitions. The repetition number of 3 is rather low, but due to time and computational constraints we were not able to increase the experiment size, therefore the interpretation should be treated with caution.

Performance Analysis of AUMVC with OC-SVM Table 14 shows, that for the *perc5* data it seems like using the default setting of OC-SVM_{cv} already yields high AUC, and applying T_{AUMVC} is worse than using the default method with respect to the AUC. The same applies for the *mammography* data, although one experiment yields the optimal value of 0 for tuning measure AUMVC. However, when executing the experiment with the *annthyroid* data, using T_{AUMVC} leads to an improvement of the AUC value in comparison to using the default setting of OC-SVM_{cv}.
Regarding the performing behaviour with respect to the number of Monte-Carlo samples, it seems like there is no pattern. When increasing the Monte-Carlo samples for T_{AUMVC}, the AUC performance tends to decrease (in the mean) when applying on the *mammography* data and to increase (in the mean) on the *annthyroid* data. In this experiment, it doesn't seem like the tuning performance with T_{AUMVC} depends on the ratio between the number of Monte-Carlo Sample size and data size. The goodness of AUMVC appears to depend on internal characteristics of the underlying data or other factors.

OC-SVM - perc5 (n = 1050, d = 4)

Mean of Repetition

Tuning Measurements		Additional Measurements			Runtime
/		auc = 1	10 n.sim = 92.56	100 n.sim = 81.09	
			1000 n.sim = 80.74	10000 n.sim = 81.42	0.39 s
auc	1		10 n.sim = 107.41	100 n.sim = 106.94	
			1000 n.sim = 106.4	10000 n.sim = 132.42	3.52 min
10 n.sim	69.34	auc = 0.95			0.39 min
100 n.sim	72.43	auc = 0.92			0.38 min
1000 n.sim	66.07	auc = 0.89			0.56 min
10000 n.sim	73.67	auc = 0.98			2.30 min

OC-SVM - mammography (n = 7595, d = 6)

Mean of Repetition

Tuning Measurements		Additional Measurements			Runtime
/			10 n.sim = 0	100 n.sim = 1.83	
		auc = 0.80	1000 n.sim = 24.29	10000 n.sim = 46.57	0.7 s
auc			10 n.sim = 1929.53	100 n.sim = 106.94	
	0.87		1000 n.sim = 292.8	10000 n.sim = 321.34	23.34 min
10 n.sim	590.48	auc = 0.79			18.66 min
100 n.sim	0	auc = 0.78			18.30 min
1000 n.sim	4.45	auc = 0.76			21.46 min
10000 n.sim	28.49	auc = 0.74			33.30 min

OC-SVM - annthyroid (n = 7062, d = 6)

Mean of Repetition

Tuning Measurements		Additional Measurements			Runtime
/			10 n.sim = 136.91	100 n.sim = 140.15	
		auc = 0.62	1000 n.sim= 240.14	10000 n.sim = 281.14	2.7 s
auc			10 n.sim = 7574.40	100 n.sim = 7521.60	
	0.82		1000 n.sim = 7520.	10000 n.sim = 7502.43	22.69 min
10 n.sim	643.51	auc = 0.67			13.87 min
100 n.sim	240.43	auc = 0.66			14.16 min
1000 n.sim	63.57	auc = 0.69			16.86 min
10000 n.sim	107.46	auc = 0.69			34.11 min

Table 14: Table of the mean performance results generated by OC-SVM_{cv} (mean over 3 repetitions from Table 32). Each block contains the results from one data set (*perc5*, *mammography*, *annthyroid*) and shows the runtime as well as the value of the tuning measurement and additional measurements. The abbreviation *10 n.sim* stands for AUMVC with 10 Monte-Carlo samples.

Performance Analysis of AUMVC with LOF For the experiments with LOF_{cv}, we can only use AUMCV with 1000 and 10000 Monte-Carlo samples. The reason is that the parameter k, which indicates the number of neighbors in the LOF method, has to

be smaller than the data size. When calculating the AUMVC, LOF_{cv} is applied to a subsample of size *n.sim* (number of Monte-Carlo samples) drawn from a hypercube (see Section 3.2 for more details), therefore $n.sim \geq k$ is required.
Table 15 shows that using LOF on the *perc5* data leads to a mean AUC of value 1 when using T_{AUMVC} instead of T_0, which is as good as T_{AUC}. Using LOF on the *mammography* data with T_{AUMVC} leads to worse mean AUC (0.69 (for T_{AUMVC} with 1000 Monte-Carlo samples) 0.68 (for T_{AUMVC} with 10000 Monte-Carlo samples)) values than using the default setting of LOF (AUC = 0.72). Applying T_{AUC} on *annthyroid* data seems to be as good as T_0 (AUC = 0.73), therefore the potential to improve when using T_{AUMVC} is already rather small. Nevertheless, in one experiment the AUC value for T_{AUMVC} with Monte-Carlo sample size of 1000 (*1000 n.sim*) is higher (AUC = 0.74) than the AUC value when applying T_{AUC}. This might be a coincidence and can be subject for further research.

LOF - perc5 (n = 1050, d = 4)

Mean of Repetition

Tuning Measurements		Additional Measurements			Runtime
\		auc = 0.92	1000 n.sim = 436.83	10000 n.sim = 436.82	2 s
auc	1		1000 n.sim = 430.25	10000 n.sim = 430.27	4.23 min
1000 n.sim	426	auc = 1			2.65 min
10000 n.sim	435.13	auc = 1			16.53 min

LOF - mammography (n = 7595, d = 6)

Mean of Repetition

Tuning Measurements		Additional Measurements			Runtime
\		auc = 0.72	1000 n.sim = 99056.26	10000 n.sim = 99060.33	3 s
auc	0.77		1000 n.sim = 85407.18	10000 n.sim = 85408.26	12 min
1000 n.sim	83494.28	auc = 0.69			9.39 min
10000 n.sim	97111.35	auc = 0.68			24.05 min

LOF - annthyroid (n = 7062, d = 4)

Mean of Repetition

Tuning Measurements		Additional Measurements			Runtime
\		auc = 0.73	1000 n.sim = 339035.88	10000 n.sim = 339041.08	3 s
auc	0.73		1000 n.sim = 350376.81	10000 n.sim = 350376.81	11.40 min
1000 n.sim	359712.75	auc = 0.74			8.69 min
10000 n.sim	321367.43	auc = 0.73			22.22 min

Table 15: Table of the mean performance results generated by LOF_{cv} (mean over 3 repetitions from Table 33). Each block contains the results from one data set (*perc5*, *mammography*, *annthyroid*) and shows the runtime as well as the value of the tuning measurement and additional measurements. The abbreviation *10 n.sim* stands for AUMVC with 10 Monte-Carlo samples.

Runtime Analysis with Tuning Using T_{AUC} takes time due to threshold tuning, which is done after each hyper parameter evaluation. The AUMVC doesn't need to tune

the threshold as it doesn't use the confusion matrix or any labels, therefore tuning with a number of Monte-Carlo samples up to 1000 is faster than AUC, but tuning with 10000 or higher is slower (see Tables 32, 33, 14 and 15). However, the runtime highly depends on the data set and the method as well.

As there is no obvious best choice for the number of Monte-Carlo samples in terms of performance, we will use $n.sim = 1000$ to reduce computational time for further experiments.

9.1.4 AUMVC(hd) Performance w.r.t. Number of Splits of the Alpha Interval

The same analysis as for the number of Monte-Carlo samples is executed for the number of alpha splits. The AUMVC approximates the area under the mass volume curve by dividing the area into trapezes. The number of trapezes under the curve is set by the number of alpha splits, which is the hyper parameter *n.alpha*. It is to be expected that with an increasing number of alpha splits (the more trapeze were used to approximate the area), the more accurate the curve is approximated and therefore the better is the performance measurement.
In contrast to this expectation, when executing the analysis with $n.alpha \in \{50, 100, 150\}$, the results in Table 34 and Table 35 and the mean results in Table 16 and Table 17 show, that there is no pattern of improvement or worsening when increasing the number of alpha splits.
For $n.alpha = 50$, 12 experiments out of 18 yield the best or at least equal AUC value in comparison to other *n.alpha* settings (compare Tables 34 and 35), for $n.alpha = 100$, 9 experiments and 6 experiments for $n.alpha = 150$. Although 18 experiments is not a sufficient number and there is no clear best *n.alpha* value, future experiments are using $n.alpha = 50$.

OC-SVM - perc5 (n = 1050. d = 4)

Mean of Repetition

Tuning Measurement		Additional Measurement			Runtime
/		auc = 1	50 n.alpha = 82.95	100 n.alpha = 82.61	0.49 s
			150 n.alpha = 81.72		
auc	1		50 n.alpha = 104.94	100 n.alpha = 104.21	5.87 min
			150 n.alpha = 103.88		
50 n.alpha	71.18	auc = 0.94			46.64 s
100 n.alpha	75.39	auc = 0.85			47.51 s
150 n.alpha	73.91	auc = 0.93			49.85 s

OC-SVM - mammography (n = 7595. d = 6)

Mean of Repetition

Tuning Measurement		Additional Measurement			Runtime
/		auc = 0.80	50 n.alpha = 40.30	100 n.alpha = 56.21	2.74 s
			150 n.alpha = 23.74		
auc	0.87		50 n.alpha = 278.2	100 n.alpha = 325.27	28.28 min
			150 n.alpha = 258.86		
50 n.alpha	4.86	auc = 0.73			20.96 min
100 n.alpha	9.04	auc = 0.81			20.44 min
150 n.alpha	2.57	auc = 0.77			18.19 min

OC-SVM - annthyroid (n = 7062. d = 6)

Mean of Repetition

Tuning Measurement		Additional Measurement			Runtime
/		auc = 0.62	50 n.alpha = 414.76	100 n.alpha = 263.48	1.83 s
			150 n.alpha = 219.33		
auc	0.81		50 n.alpha = 4130.89	100 n.alpha = 4931.88	19.18 min
			150 n.alpha = 3878.36		
50 n.alpha	151.5	auc = 0.74			17.00 min
100 n.alpha	161.62	auc = 0.69			16.78 min
150 n.alpha	157.54	auc = 0.64			16.13 min

Table 16: Table of the mean performance results generated by OC-SVM (mean over three repetitions from Table 34). Each block contains the results from one data set (*perc5*, *mammography*, *annthyroid*) and shows the runtime as well as the value of the tuning measurement and additional measurements. The abbreviation *50 n.alpha* stands for AUMVChd with 50 internal subsamples evaluated with AUMVC.

LOF - perc5 (n = 1050, d = 4)

Mean of Repetition

Tuning Measurement		Additional Measurement			Runtime
/		auc = 0.92	50 n.alpha = 436.77 150 n.alpha = 442.89	100 n.alpha= 441.35	2.09 s
auc	1		50 n.alpha = 433.83 150 n.alpha = 439.94	100 n.alpha = 438.41	8.52 min
50 n.alpha	433.77	auc = 1			6.53 min
100 n.alpha	438.3	auc = 1			6.19 min
150 n.alpha	439.87	auc = 1			6.28 min

LOF - mammography (n = 7595, d = 6)

Mean of Repetition

Tuning Measurement		Additional Measurement			Runtime
/		auc = 0.72	50 n.alpha= 99066.98 150 n.alpha = 100449.14	100 n.alpha = 100102.91	3.75 s
auc	0.77		50 n.alpha = 99069.59 150 n.alpha = 100454.58	100 n.alpha= 100111.77	56.35 min
50 n.alpha	99044.17	auc = 0.68			53.07 min
100 n.alpha	100110.9	auc = 0.70			51.25 min
150 n.alpha	100423.87	auc = 0.67			53.11 min

LOF - annthyroid data (n = 7062, d = 4)

Mean of Repetition

Tuning Measurement		Additional Measurement			Runtime
/		auc = 0.73	50 n.alpha = 339038.56 150 n.alpha = 343773.78	100 n.alpha = 342602.16	5.36 s
auc	0.74		50 n.alpha = 339041.08 150 n.alpha = 343777.40	100 n.alpha = 342608.44	27.11 min
50 n.alpha	339014.56	auc = 0.72			17.37 min
100 n.alpha	342608.44	auc = 0.72			15.35 min
150 n.alpha	343781.59	auc = 0.71			15.38 min

Table 17: Table of the mean performance results generated by LOF (mean over three repetitions from Table 35). Each block contains the results from one data set (*perc5*, *mammography*, *annthyroid*) and shows the runtime as well as the value of the tuning measurement and additional measurements. The abbreviation *50 n.alpha* stands for AUMVChd with 50 internal subsamples evaluated with AUMVC.

9.1.5 AUMVChd Performance w.r.t. to Number of Internal Subsamples

The aim of this analysis is to investigate if the quality of the performance measurement AUMVChd improves with increasing internal feature subsamples (*amv.iters*).
For using the MV curve to evaluate the performance of methods applied to high dimensional data ($d > 8$) (*perc5.hd* and *penlocal*), we need to use AUMVChd instead of AUMVC. As described in Section 3.2, AUMVChd repeats AUMVC on *amv.iters* subsamples of the data. The subsamples have smaller dimensions of size *amv.feats* (≤ 5) and less observations of size *n.sim* than the passed data.
In this subanalysis we set $amv.feats = 5$ as a fixed hyper parameter and only vary the

number of internal subsamples *amv.iters*. The larger *amv.iters*, the larger is the probability that every feature from the original passed data set is contained in at least one subsample.
The results in Table 18 and 19 don't show a pattern for increasing tuning performance when increasing the number of internal subsamples. When having a closer look at the detailed Tables 36 and Table 37, it can be derived that *20 amv.iters* yields the best or at least equal AUC value in comparison to other *amv.iters* settings in 9 from 12 experiments. *50 amv.iters* only yields as best performance in 2 examples, and *5 amv.iters* in 6 examples.
Based on these results, we will use *amv.iters* = 20 for further experiments.

OC-SVM - perc5 (n = 1050, d = 10)

Mean of Repetition

Tuning Measurements		Additional Measurements			Runtime
/		auc = 1	5 amv.iters = 765.27 50 amv.iters = 749.01	20 amv.iters = 758.09	7 s
auc	1		5 amv.iters = 763.84 50 amv.iters = 756.94	20 amv.iters = 768.33	32 min
5 amv.iters	805.30	auc = 0.95			5.12 min
20 amv.iters	766.69	auc = 1			11.4 min
50 amv.iters	744.21	auc = 0.94			23.56 min

OC-SVM - penlocal (n = 6724, d = 16)

Mean of Repetition

Tuning Measurements		Additional Measurements			Runtime
/		auc = 0.48	5 amv.iters = 18.65 50 amv.iters = 19.19	20 amv.iters = 19.43	44 s
auc	0.97		5 amv.iters = 19.59 50 amv.iters = 18.84	20 amv.iters = 19.37	9.94 min
5 amv.iters	20.30	auc = 0.70			21.81 min
20 amv.iters	18.67	auc = 0.94			49.88 min
50 amv.iters	18.55	auc = 0.52			1 h

Table 18: Table of the mean performance results generated by OC-SVM (mean over three repetitions in Table 36). Each block contains the results from one data set (*perc5.hd*, *penlocal*) and shows the runtime as well as the value of the tuning measurement and additional measurements. The abbreviation *5 amv.iters* stands for AUMVChd with 5 internal iterations of applying AUMVC on subsamples.

LOF - perc5 (n = 1050, d = 10)

Mean of Repetition

Tuning Measurements		Additional Measurements			Runtime
/		auc = 1	5 amv.iters = 3094.71	20 amv.iters = 3137.50	21 s
			50 amv.iters = 3072.48		
auc	0.92		5 amv.iters = 3033.43	20 amv.iters = 3110.28	1 min
			50 amv.iters = 3088.06		
5 amv.iters	3100.88	auc = 0.93			14.24 min
20 amv.iters	3043.77	auc = 0.87			45.06 min
50 amv.iters	3188.71	auc = 1			2.07 h

LOF - penlocal (n = 6724, d = 16)

Mean of Repetition

Tuning Measurements		Additional Measurements			Runtime
/		auc = 0.98	5 amv.iters = 38.97	20 amv.iters = 44.61	1.95 min
			50 amv.iters = 41.18		
auc	0.97		5 amv.iters = 40.86	20 amv.iters = 43.38	11.33 min
			50 amv.iters = 42.79		
5 amv.iters	46.72	auc = 0.94			1.29 h
20 amv.iters	35.01	auc = 0.92			2.8 h
50 amv.iters	40.27	auc = 0.90			4.63 h

Table 19: Table of the mean performance results generated by OC-SVM (mean over three repetitions from Table 37). Each block contains the results from one data set (*perc5.hd*, *penlocal*) and shows the runtime as well as the value of the tuning measurement and additional measurements. The abbreviation *5 amv.iters* stands for AUMVChd with 5 internal iterations of applying AUMVC on subsamples.

9.2 Performance Analysis under Different Experiments Setups

After conducting an extensive analysis of the settings for the unsupervised measurements AUMVC and AUMVChd, this section will further investigate the OC-SVM and LOF methods, the two different resampling types R_{cv} n and R_{occv} as well as the performance of AUMVC and AUMVChd on different low- and high dimensional data sets.
The analysis of the AE_{cv} and AE_{occv} is done separately in Section 9.2.4. , due to computational challaenges.
The parameter settings for the AUMVC(hd) are stated in Table 7 and the tuning settings for the benchmark experiments are in Table 6. As mentioned in Section 8.2, we always train on data without labels (unsupervised setting), but test on data with labels when using T_{AUC} and test on data without labels when using T_{AUMVC} or $T_{AUMVC_{hd}}$. For the final evaluation, we calculate the AUC and AUMVC(hd) values.
There are 18 experiments for each AUMVC on low dimensional data, and AUMVChd on high dimensional data. The Tables 38 to 49 in the Appendix are the numerical results tables for all following sections. The Figures 19 and 18 visualise the results

using AUMVC on low dimensional data sets, Figure 21 and 20 visualise the results using AUMVChd on high dimensional data sets. Three aspects of these figures are analysed:

1. Comparing T_0, T_{AUC} and $T_{AUMVC}/T_{AUMVChd}$ (Section 9.2.1).
2. Comparing the effect of using R_{cv} and R_{occv} on OC-SVM and of using T_{AUC}, T_{AUMVC}, $T_{AUMVChd}$ (Section 9.2.2).
3. Comparing the methods LOF with OC-SVM (Section 9.2.3).

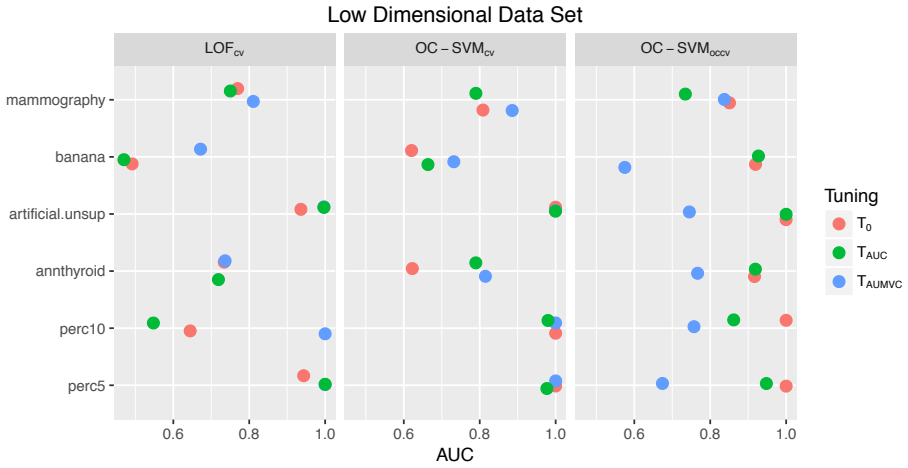


Figure 18: Aggregated AUC performance of the nested resampling results on low dimensional dats sets, when using OC-SVM_{cv}, OC-SVM_{occv}, LOF_{cv} combined with T_{AUC} and T_{AUMVC} as well as T_0. The figure is grouped by resampling-method combination, the left panel shows the results calculated with LOF_{cv}, the middle panel with OC-SVM_{cv}, the right with OC-SVM_{occv}.

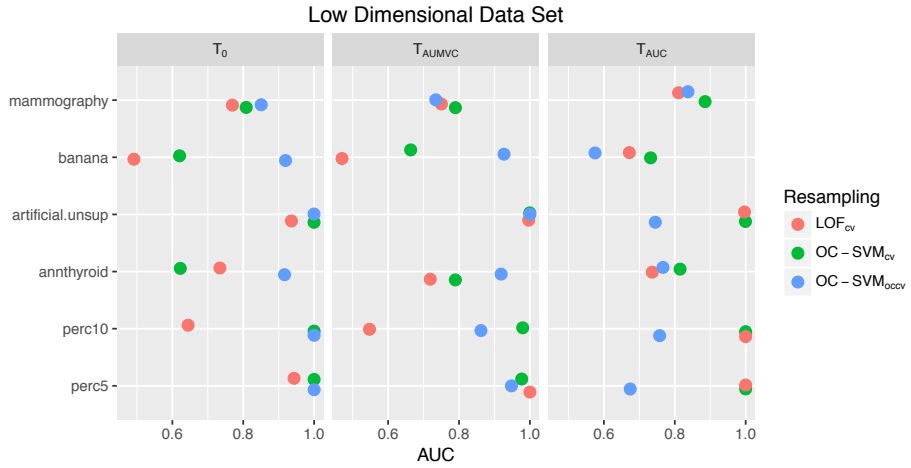


Figure 19: Aggregated AUC performance of the nested resampling results on low dimensional data sets, when using OC-SVM_{cv}, OC-SVM_{occv}, LOF_{cv} combined with T_{AUC} and T_{AUMVC} as well as T_0. The figure is grouped by tuning types, the left panel shows the results for T_0, the middle panel shows results for T_{AUMVC}, on the right panel shows the results for T_{AUC}.

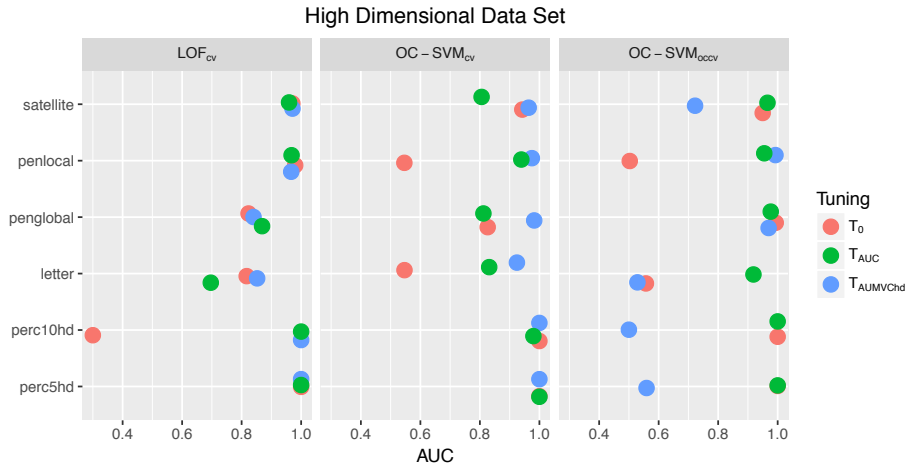


Figure 20: Aggregated AUC performance of the nested resampling results on high dimensional dats sets, when using OC-SVM_{cv}, OC-SVM_{occv}, LOF_{cv} combined with T_{AUC} and $T_{AUMVChd}$ as well as T_0. The figure is grouped by resampling-method combination, the left panel shows the results calculated with LOF_{cv}, the middle panel with OC-SVM_{cv}, the right with OC-SVM_{occv}.

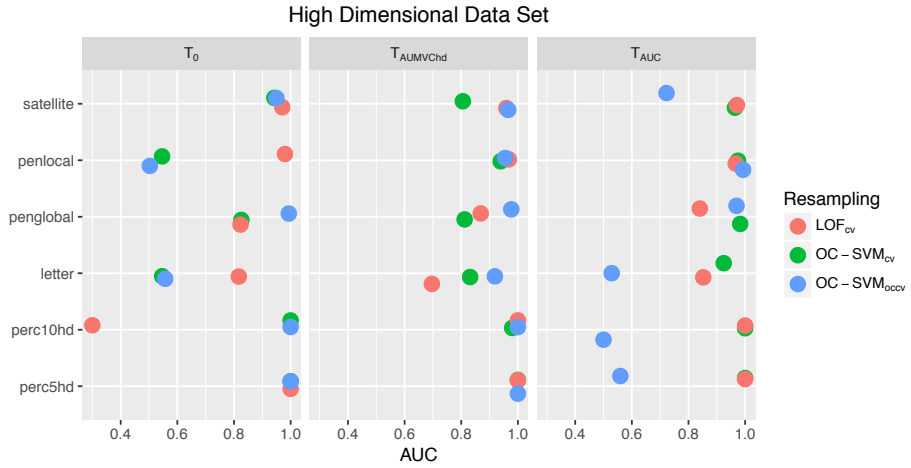


Figure 21: Aggregated AUC performance of the nested resampling results on high dimensional data sets, when using OC-SVM_{cv}, OC-SVM_{occv}, LOF_{cv} combined with T_{AUC} and $T_{AUMVChd}$ as well as T_0. The figure is grouped by tuning cases, the left panel shows the results for T_0, the middle panel shows results for $T_{AUMVChd}$, the right panel shows the results for T_{AUC}.

9.2.1 Tuning with Tuning Measurement AUMVC and AUMVChd

This Section is about analysing if AUMVC(hd) is a good alternative measurement for AUC when tuning without available labels and a good alternative compared to not tuning at all. Although in some experiments T_0 performs better than T_{AUC}, we assume that having labels and T_{AUC} is the best case scenario and not having labels and using the default settings of the applied method is the worst case scenario.

9.2.1.1 AUMVC on Low Dimensional Data

If we don't take into account that the experiments are generated by the different methods and resampling strategies and only look at the outputs of AUC and AUMVC, we can summarise the results as in Table 20 and visualised in Figure 22. Each row of Table 20 contains 18 comparisons of experiments. Thus the number of experiments for which the statements in the row are true has to be interpreted relatively to the 18 comparisons.
In 10 examples T_{AUMVC} was worse than T_0, but in 8 examples there is no worsening when using T_{AUMVC}, as the resulting AUC value is at least as high as the AUC value when applying T_{AUC} (best case scenario) and in 5 examples the performance was even better than T_0. Additionally, in 8 examples T_{AUMVC} was at least as good as T_{AUC}, and in 5 examples T_{AUMVC} even outperformed T_{AUC}. However, all in all it appears as if it should not be recommended to tune with AUMVC.
In terms of runtime T_{AUMVC} performance was better, as it doesn't take as much time as T_{AUC}. The reason is, that AUC additionally requires threshold tuning. Taking the mean of runtime over all experiments within the cases T_{AUC} and T_{AUMVC} shows that

T_{AUC} takes nearly twice as long as the other cases. On average, T_{AUC} runs about 9.75 minutes (extract from Table 40) and T_{AUMVC} about 5.26 minutes (see from Table 41). Figure 22 reveals that tuning with T_{AUC} does not always improve the performance in comparison to the default settings in T_0, which is also the case for tuning without labels T_{AUMVC}. It also provides insight on how far the AUC from T_{AUC} and T_{AUMVC} is away from T_0 (Figure 22 left panel) as well as T_0 and T_{AUC} from the AUC from T_{AUMVC} (Figure 22 right panel). The left panel shows that if using T_{AUC} and if the model underperforms in comparison to T_0, it underperforms more strongly than in case of using T_{AUMVC}, but when T_{AUC} overperforms in comparison to T_0 it outperforms using T_{AUMVC} more strongly. All in all, in most of the T_{AUMVC} examples the AUC lies within the ± 0.1 range of T_0.
A look at the plot on the right hand side of Figure 22 shows that the ideal case is to have the AUC of T_0 on the left of the benchmark line and the AUC of T_{AUC} on the right or on the benchmark line is not met, but in some examples using T_{AUMVC} outperforms using T_{AUC} by over 0.2 points.
The middle panel in Figure 19 shows that in almost all cases using T_{AUMVC} with OC-SVM_{cv} yields at least an AUC value of 0.8, which reveals better results than combining T_{AUMVC} with OC-SVM_{CV} or LOF_{cv}.

Using T_{AUMVC} yields AUC value ...	Number of Experiments (18 comparisons for each row)
$<$ AUC of T_0	10
$=$ AUC of T_0	3
$>$ AUC of T_0	5
$<$ AUC of T_{AUC}	10
$=$ AUC of T_{AUC}	3
$>$ AUC of T_{AUC}	5

Table 20: Summary of how the AUC measures perform under different tuning settings (T_0,T_{AUC},T_{AUMVC}) on low dimensional data, independent of the method used. Note: The distribution of case T_0 is equal to case T_{AUC} in this analysis.

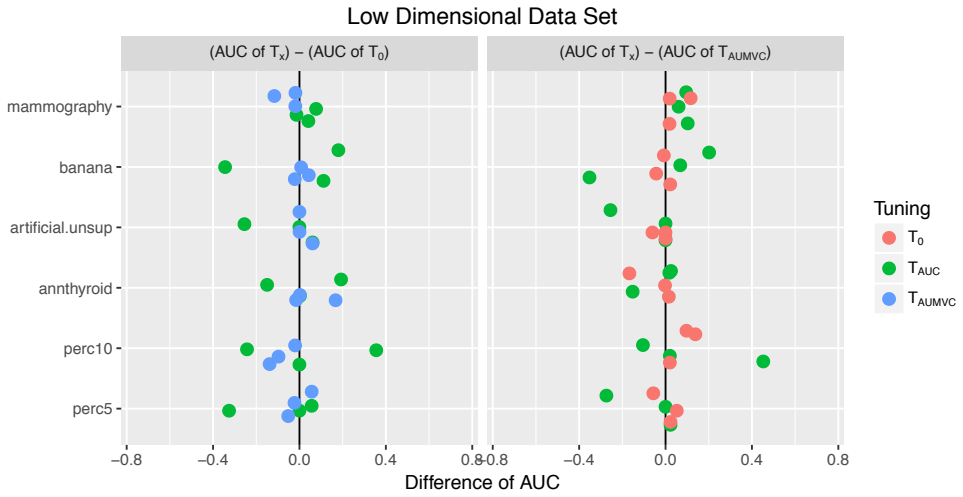


Figure 22: Left panel: visualissation of the difference between the AUC resulting from T_{AUC} and T_{AUMVC} and the AUC resulting from T_0 (= vertical benchmark line). Ideally, all examples lie on the right hand side of the benchmark line.
Right panel: Visualisation of the difference between the AUC resulting from T_0 and T_{AUC} and the AUC resulting from T_{AUMVC} (= vertical benchmark line). Ideally, the AUC value of T_0 is lower than the AUC of T_{AUMVC} (all red points on left side of the benchmark line), and the AUC value of T_{AUC} is higher or equal to the AUC of T_{AUMVC} (all green points on the right side or on the benchmark line).
All settings are applied on low dimensional data sets.

9.2.1.2 AUMVChd on High Dimensional Data

The same analysis as for AUMVC is conducted for AUMVChd on high dimensional data. The results are summarised in Table 21 and Figure 23. As in the case before, 18 experiments is the reference number for each row in Table 21.
In 7 examples using $T_{AUMVChd}$ performs worse than T_0. In 4 examples the performance of T_0 and T_{AUMVC} are equal. In 7 examples $T_{AUMVChd}$ performed better compared to T_{AUC}, and in 6 examples T_{AUMVC} outperformed T_{AUC}.
However, when it comes to runtime, $T_{AUMVChd}$ needs more time than T_{AUC}. Taking the runtime mean over all examples for the cases T_{AUC} and T_{AUMVC} shows that for the former it takes 2.4 minutes on average and for the latter about 1h. (Note: The absolute runtime in this section on high dimensional data can't be compared to the runtime of the previous section on low dimensional data, as it was executed on different machines.)
Figure 23 for high dimensional data is more scattered than the Figure 22 for low dimensional data. The left panel shows that T_{AUMVC} in comparison to T_0 is better than T_{AUC} in comparison to T_0. The right panel shows that using T_{AUMVC} is outperformed by a maximum of 0.2 points by T_{AUC} or T_0, but outperform T_{AUC} or T_0 up to 0.75 points .
All in all, it appears that using $T_{AUMVChd}$ instead of T_{AUC} on high dimensional data

is a better alternative than using T_{AUMVC} instead of T_{AUC} on low dimensional data in terms of performance measure AUC.
The middle panel in Figure 21 shows that combining $T_{AUMVChd}$ with OC-SVM_{cv} yields high AUC performance in comparison to the other combinations, as all examples have an AUC of at least 0.9.

Using T_{AUMVC} yields AUC value ...	Number of Experiments (18 comparisons for each row)
$<$ AUC of T_0	7
$=$ AUC of T_0	4
$>$ AUC of T_0	7
$<$ AUC of T_{AUC}	8
$=$ AUC of T_{AUC}	4
$>$ AUC of T_{AUC}	6

Table 21: Summary of how the AUC measure performs under different tuning settings (T_0,T_{AUC},T_{AUMVC}) on low dimensional data, independent of the method or resampling strategy.

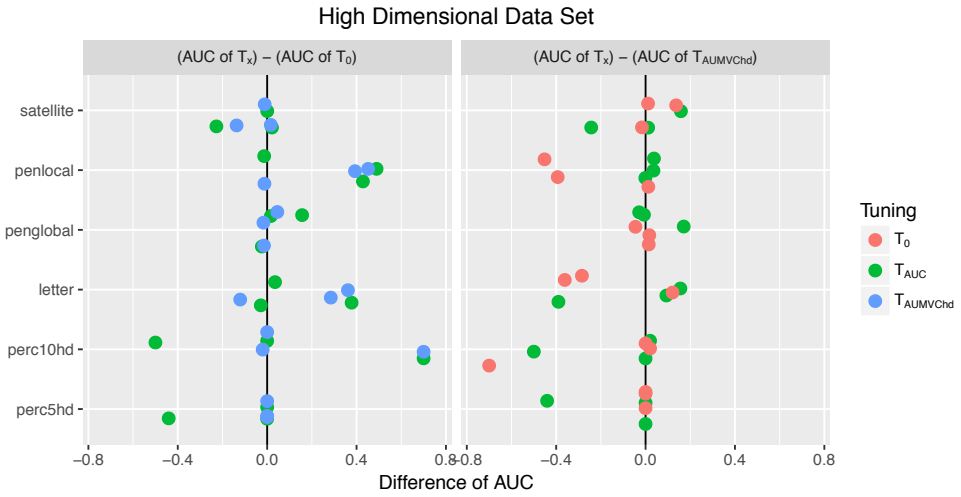


Figure 23: Left panel: Visualisation of the difference between the AUC resulting from T_{AUC}, $T_{AUMVChd}$ and the AUC resulting from T_0 (= vertical benchmark line). Ideally, all examples lie on the right hand side of the benchmark line. Right panel: Visualisation of the difference between the AUC resulting from T_0, T_{AUC} and the AUC resulting from $T_{AUMVChd}$ (= vertical benchmark line). Ideally, the AUC value of T_0 is lower than the AUC of $T_{AUMVChd}$ (all red points on left side of the benchmark line), and the AUC value of T_{AUC} is higher or equal to the AUC of $T_{AUMVChd}$ (all green points on the right side or on the benchmark line).
All settings are applied on high dimensional data sets.

9.2.2 Training and Tuning with Cross-Validation vs. One-Class Cross-Validation for OC-SVM

The OC-SVM method is designed to train on data containing only one class (the normal class). Therefore, we introduced the R_{occv} resampling method in Section 5. Since in most real-life data sets it is not guaranteed to only have normal data in training, the experiment with OC-SVM_{cv} is included. The standard R_{cv} doesn't take into account the labels, thus it creates training data sets containing both classes (normal and anomaly). We want to investigate how the violation of the one class assumptions affect the OC-SVM method, which is why this section excludes the LOF method.

9.2.2.1 AUMVC on Low Dimensional Data

When comparing the resampling strategies R_{cv} and R_{occv} with OC-SVM on low dimensional data sets, Table 22, Figure 19 and Figure 24 give more insights about the AUC performance. If we don't take into account that the experiments are generated by different tuning measures but only look at the outputs of AUC and AUMVC, we can

summarise the results of how AUC performs when using R_{occv} instead of R_{cv} to the first column in Table 22 (each row in this column contains 18 comparisons). If we do take the different tuning measures into account, the results can be summarised to the last three columns in Table 22 (each row in each of those columns contains 6 comparisons). In 13 from 18 examples the OC-SVM performs with R_{cv} at least as good as with R_{occv} (Table 22 or Figure 24), which indicates, that even if the requirement of OC-SVM to train on only one class is not fulfilled, OC-SVM_{cv} still performs well or even better. The most realistic case in Table 22 or Figure 24 is to use T_{AUMVC} with R_{cv}, and in 4 out of 6 examples using R_{cv} performs at least as good as with R_{occv}, but the negative effect for the other two examples is up to 0.3 AUC points.
In real life, the assumption of only having one class in training is often violated, this results tend to reassure that OC-SVM can still be used for tuning in that case. However, not tuning OC-SVM_{cv} is not recommended, based on these examples. It should be noticed that 6 experiments might not be representative.

Applying OC-SVM_{cv} yields AUC value	Over all Tuning Cases (each row 18 comparisons)	within T_0 (each row 6 comparisons)	within T_{AUC} (each row 6 comparisons)	within $T_{AUMVChd}$ (each row 6 comparisons)
$<$ AUC value of OC-SVM_{occv}	$\sum 5$	3	0	2
$=$ AUC value of OC-SVM_{occv}	$\sum 4$	3	0	1
$>$ AUC value of OC-SVM_{occv}	$\sum 9$	0	6	3

Table 22: Summary of performance results when using OC-SVM_{cv} instead of OC-SVM_{occv} based on 6 different low dimensional data sets and tuning setting T_0, T_{AUC} and $T_{AUMVChd}$.

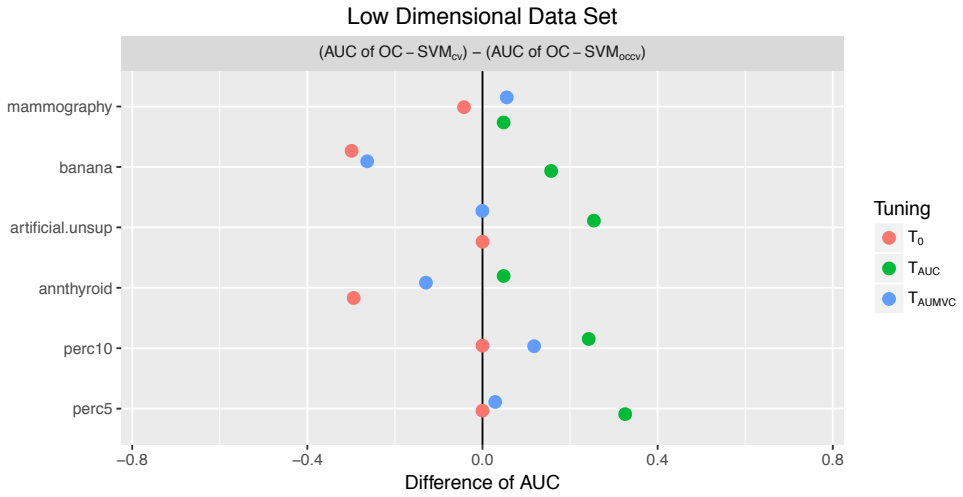


Figure 24: Visualisation of the difference between the AUC resulting from OC-SVM_{cv} and the AUC resulting from OC-SVM_{occv} (= vertical benchmark line) applied on low dimensional data sets. Examples on the right hand side of the benchmark line indicate that using OC-SVM_{cv} outperforms OC-SVM_{occv}. Ideally, all examples should be on the right or on the benchmark line. In that case the requirement violation of OC-SVM training on only one-class doesn't matter.

9.2.2.2 AUMVChd on High Dimensional Data

When analysing the effect of resampling strategy R_{cv} and R_{occv} in combination with OC-SVM on high dimensional data, the effects appear to be slightly contradictory to the low dimensional analysis (Figure 21, Figure 25, Table 23). If we don't take into account that the experiments are generated by different tuning measures, but only look at the outputs of AUC and AUMVChd, we can summarise the results of how AUC performs when using R_{occv} instead of R_{cv} to the first column in Table 23 (each row in this column contains 18 comparisons). If we do take the different tuning measures into account, the results can be summarised to the last three columns in Table 23 (each row in each of those columns contain 6 comparisons) or Figure 25.
Table 23 and Figure 25 show that in 9 out of 18 examples violation of training on one-class only doesn't matter (at least as good as OC-SVM_{occv}), but at the same time in 9 examples it does negatively impact the performance of OC-SVM. Especially, in the most realistic case of T_{AUMVC} only 1 out of 6 examples in Table 23 or Figure 25 show that using R_{cv} instead of R_{occv} doesn't matter. Figure 25 also shows that the requirement violations affect the OC-SVM negatively by only up to 0.2 AUC points. When applying on high dimensional data, the violation negatively affects the performance of OC-SVM. The same effect applies for T_0. It should be noted that 6 experiments might not be representative.

Applying OC-SVM_{cv} yields AUC value	Over all Tuning Cases (each row 18 comparisons)	within T_0 (each row 6 comparisons)	within T_{AUC} (each row 6 comparisons)	within $T_{AUMVChd}$ (each row 6 comparisons)
$<$ AUC value of OC-SVM_{occv}	$\sum$ 9	3	1	5
$=$ AUC value of OC-SVM_{occv}	$\sum$ 3	2	0	1
$>$ AUC value of OC-SVM_{occv}	$\sum$ 6	1	5	0

Table 23: Summary of performance results when using OC-SVM_{cv} instead of OC-SVM_{occv} based on 6 different high dimensional data sets and tuning setting T_0, T_{AUC} and $T_{AUMVChd}$.

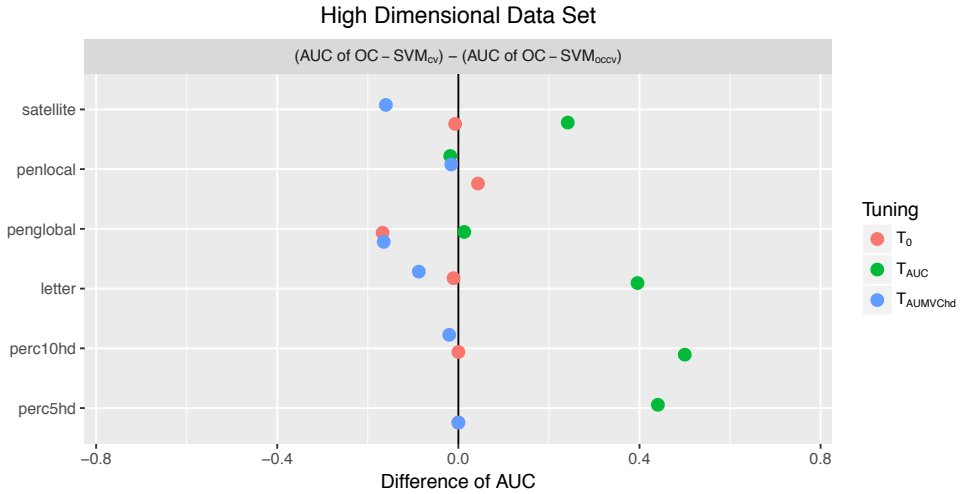


Figure 25: Visualisation of the difference between the AUC resulting from OC-SVM_{cv} and the AUC resulting from OC-SVM_{occv} (= vertical benchmark line), applied on high dimensional data sets. Examples on the right hand side of the benchmark line indicate that using OC-SVM_{cv} outperforms OC-SVM_{occv}. Ideally, all examples should be on the right or on the benchmark line, because it would imply that violation of the requirement doesn't affect the performance of the method.

9.2.3 Comparison of Methods OC-SVM and LOF

In Section 9.2.2 OC-SVM was compared to the case when using R_{cv} or R_{occv}, whereas in this section both OC-SVM cases will be compared to LOF in general and grouped by tuning cases T_0, T_{AUC} and T_{AUMVC}. Although LOF_{cv} should only be compared to OC-SVM_{cv} as R_{cv} allows mixed trainng data, this section additionally includes the comparison of LOF_{cv} to OC-SVM_{occv}.

9.2.3.1 AUMVC on Low Dimensional Data

Comparing LOF_{cv} to OC-SVM_{cv} in Table 24 and Figure 26 without considering different tuning settings (first column) reveals that OC-SVM_{cv} outperforms LOF_{cv} in 12 out of 18 examples in the AUC performance. In 4 examples both methods perform equal and only in 2 examples LOF_{cv} yields a higher AUC. Figure 26 clearly shows, that almost all examples for OC-SVM_{cv} are lying on the right hand side or close to the benchmark lines (LOF_{cv}). When differentiating between tuning settings, the real-life setting T_0, T_{AUMVC} are most relevant. Within this tuning setting OC-SVM_{cv} should be preferred, too.
Although the performance of LOF_{cv} is higher when comparing to OC-SVM_{occv} instead to OC-SVM_{cv}, the ultimate conclusion is still that OC-SVM should be preferred over LOF, except for when tuning with T_{AUC}. When using T_{AUC}, the method LOF_{cv} performs better in 4 out of 6 examples, but this situation should be neglected, as AUC requires labels, which is not realistic in real life situation.

Applying LOF_{cv} yields AUC value	Over all Tuning Cases (each row 18 comparisons)	within T_0 (each row 6 comparisons)	within T_{AUC} (each row 6 comparisons)	within T_{AUMVC} (each row 6 comparisons)
< AUC value of OC-SVM_{cv}	$\sum$ 12	5	3	4
= AUC value of OC-SVM_{cv}	$\sum$ 4	0	3	1
> AUC value of OC-SVM_{cv}	$\sum$ 2	1	0	1
< AUC value of OC-SVM_{occv}	$\sum$ 11	6	2	3
= AUC value of OC-SVM_{occv}	$\sum$ 1	0	0	1
> AUC value of OC-SVM_{occv}	$\sum$ 6	0	4	2

Table 24: Summary of performance results when using LOF_{cv} instead of OC-SVM_{cv} or OC-SVM_{occv} based on 6 different low dimensional data sets and tuning settings T_0, T_{AUC} and T_{AUMVC}. Note: T_{AUC} is not always better than using T_0.

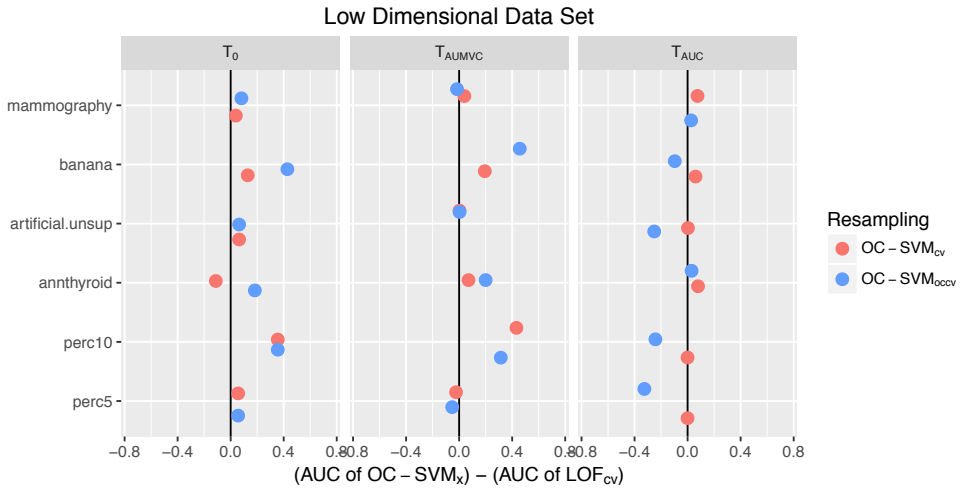


Figure 26: Visualisation of the difference between the AUC resulting from OC-SVM_{cv}, OC-SVM_{occv} and the AUC resulting from LOF_{cv} (= vertical benchmark line), applied on low dimensional data sets. The figure is grouped by tuning type, the left panel shows the results for T_0, the middle panel for $T_{AUMVChd}$ and the right panel shows the result for T_{AUC}. Examples on the right hand side of the benchmark line indicate that using OC-SVM outperforms LOF_{cv}.

9.2.3.2 AUMVChd on High Dimensional Data

When applying the three methods LOF_{cv}, OC-SVM_{cv} and OC-SVM_{occv} on high dimensional data sets, the OC-SVM methods are not as clearly favorable as when applying them on low dimensional data sets.
In fact, LOF_{cv} is at least as good as OC-SVM_{cv} in 13 out of 18 examples, of those 8 examples outperform OC-SVM_{cv}.
When comparing LOF_{cv} to OC-SVM_{occv}, the conclusion is not that clear either, overall it appears like in 50% of the cases one method outperforms the other (7 to 8 examples in Table 25). Excluding the unrealistic case of T_{AUC}, LOF_{cv} performs at least as good as OC-SVM_{occv} in 7 out of 12 examples using T_0 and $T_{AUMVChd}$.
Within the most realistic tuning cases $T_{AUMVChd}$, LOF_{cv} seems to yield an AUC close to both OC-SVM methods (Figure 27). Thus, LOF_{cv} is favored over OC-SVM on high dimensional data.

Applying LOF_{cv} yields AUC value	Over all Tuning Cases (each row 18 comparisons)	within T_0 (each row 6 comparisons)	within T_{AUC} (each row 6 comparisons)	within $T_{AUMVChd}$ (each row 6 comparisons)
$<$ AUC value of OC-SVM_{cv}	$\sum$ 5	2	2	1
$=$ AUC value of OC-SVM_{cv}	$\sum$ 5	1	3	1
$>$ AUC value of OC-SVM_{cv}	$\sum$ 8	3	1	4
$<$ AUC value of OC-SVM_{occv}	$\sum$ 7	2	2	3
$=$ AUC value of OC-SVM_{occv}	$\sum$ 3	1	0	2
$>$ AUC value of OC-SVM_{occv}	$\sum$ 8	3	4	1

Table 25: Summary of performance results when using LOF_{cv} instead of OC-SVM_{cv} or OC-SVM_{occv} based on six different high dimensional data sets and tuning setting T_0, T_{AUC} and $T_{AUMVChd}$. Note: T_{AUC} is not always better than using T_0.

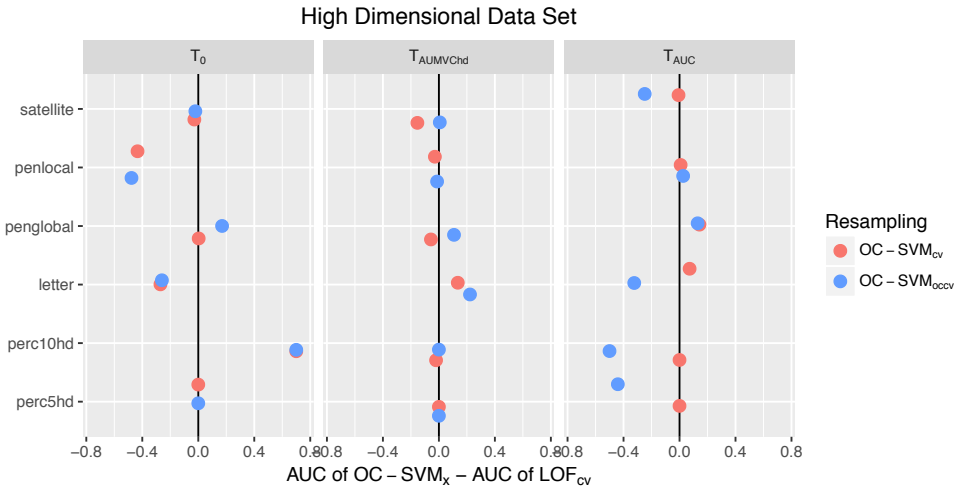


Figure 27: Visualisation of the difference between the AUC resulting from OC-SVM_{cv},OC-SVM_{occv} and the AUC resulting from LOF_{cv} (= vertical benchmark line), applied on high dimensional data sets. The figure is grouped by tuning case, the left panel shows the results for T_0, the middle panel for $T_{AUMVChd}$, the right panel shows the result for T_{AUC}. Examples on the right hand side of the benchmark line indicate that using OC-SVM outperforms LOF_{cv}.

9.2.4 Analysis of the Autoencoder

The autoencoder (AE) implemented in mlR is based on H2o (Section 7), that is not compatible with the parallel mode in mlR. Moreover, when tuning the number of hidden units in the models, they throw errors due to the prediction of an unstable model. As finding suitable hyperparameters to stable the model is a major issue, the AE was

excluded from the previous benchmark analysis and is investigated subsequently in this section.
Since the hyperparameter settings for the AE highly depends on the data set, we choose our parameter according to the H2o help page `https://0xdata.atlassian.net/browse/TN-4`, which fix some of the stability issues for our models. However, tuning on high dimensional data was still not possible. After a few iterations to find suitable hyperparameter settings we abort the experiment and analyse the autoencoder with T_0, T_{AUC}, T_{AUMVC} on the low dimensional data, T_0 on high dimensional data and T_{AUC} and T_{AUMVC} on two synthetic high dimensional data (*perc5hd*, *perc10hd*).
The autoencoder doesn't have a default for the number of hidden layers or hidden units. However, in the study of Dau et al. [2014, p.312], it is shown that an autoencoder with one hidden layer performs as good as with more. Thus, we fix our experiment to one hidden layer. There is also no rule of thumb to set the number of hidden units in the hidden layer. We therefore chose arbitrarily the number of hidden units half the size of the number of input units for the case T_0. Furthermore, we tuned the method on 4 out of 6 low dimensional data sets with OC-CV (AE_{occv}). The `banana` and `artificial.unsup` data only have two features, hence it is not reasonable to tune for the number of hidden units. For the rest of the low dimensional data we set the tuning search space according to Table 26 and reduce the random search size to 20 to save computational time, that is feasible as the range of the number of units is discrete and rather small. Due to computational time, the optimization algorithm for the high dimensional data sets *perc5hd* and *perc10hd* is set to a grid search to optimize over all 8 values (see Table 26).

9.2.4.1 AUMVC on Low Dimensional Data

The results of applying AUC_{occv} on low dimensional data are shown in Figure 26 and Figure 28, the numeric results are in Appendix Table 50. The results for high dimensional data are in Figure 31 and Figure 32 and Table 51.

Data Set	Default Value for Number of Units #Input Units : #Hidden Units : #Output Units	Tuning Search Space	Search Type
perc5	4 : 2 : 4	1 to 3	Random Search, maxit 20
perc10	4 : 2 : 4	1 to 3	Random Search, maxit 20
annthyroid	6 : 3 : 6	1 to 5	Random Search, maxit 20
artificial.nsup	2 : 1 : 2	-	Random Search, maxit 20
banana	2 : 1 : 1	-	Random Search, maxit 20
mammography	6 : 3 : 6	1 to 5	Random Search, maxit 20
perc5hd	10 : 5 : 10	1 to 8	Grid Search
perc10hd	10 : 5 : 10	1 to 8	Grid Search
letter	32 : 16 : 32	-	-
penglobal	16 : 8 : 16	-	-
penlocal	16 : 8 : 16	-	-
satellite	36 : 18 : 36	-	-

Table 26: Default and tuning settings of the number of hidden units for the AE_{occv} with one hidden layer. The settings depends on the data set.

Figure 28 visualises the results of the experiments on low dimensional data generated by AE_{occv}. It seems like tuning with T_{AUC} or T_{AUMVC} doesn't improve the performance of the AE_{occv} a lot. In fact in 3 out of 5 examples T_0 with just arbitrary chosen number of hidden units already yield a high AUC nearly 1, T_{AUC} even worsen the performance on the *annthyroid* data. On the *mammography* data, the performance improves slightly. Comparing to the other methods in Figure 29, the conclusion is that when tuning with T_{AUMVC} the AE_{occv} yields always better or same results as the other methods. The same conclusion applies wehen using T_{AUC}. In case of T_0, the OC-SVM_{occv} should be preferred, but it should also be noted that the number of nodes in the hidden layer was chosen arbitrarily.

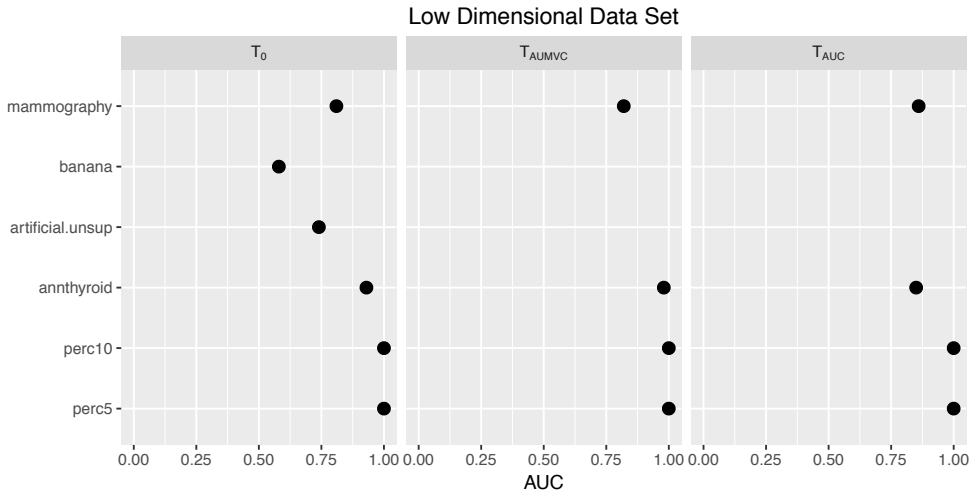


Figure 28: Aggregated AUC performance of the nested resampling results on low dimensional data sets generated by an AE_{occv}, when using T_0, T_{AUMVC} and T_{AUC}. The figure is grouped by tuning type, the left panel shows the results for T_0, the middle panel for T_{AUMVC} and the right panel shows the result for T_{AUC}.

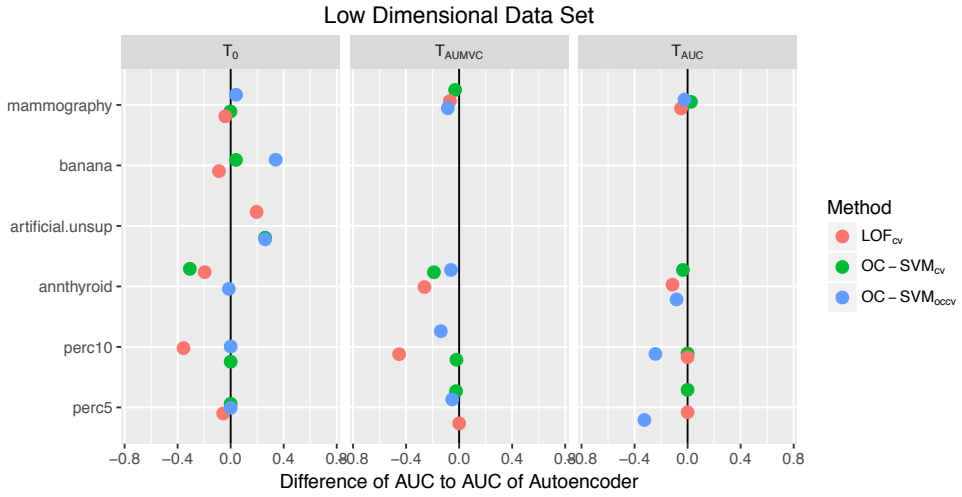


Figure 29: Visualisation of the difference between the AUC resulting from OC-SVM_{cv}, OC-SVM_{occv} LOF_{cv} and the AUC resulting from AE_{occv} (= vertical benchmark line), applied on low dimensional data sets. The figure is grouped by tuning type, the left panel shows the results for T_0, the middle panel for T_{AUMVC} and the right panel shows the result for T_{AUC}. Examples on the right-hand side of the benchmark line indicate that using AE_{occv} outperforms the other methods.

AE requires only one class in training, thus we tuned with R_{occv}. For testing if the

violation of assumptions has an affect on the AE, we also tune the low dimensional data with R_{cv}, Figure 30 and Table 52 provide the results. Using R_{cv} has a strong affect on the *annthyroid* data set, unlike the other data sets. More experiments should be conducted, but based on this example, the conclusion is that violation of the assumption of training on one class has a minor effect.

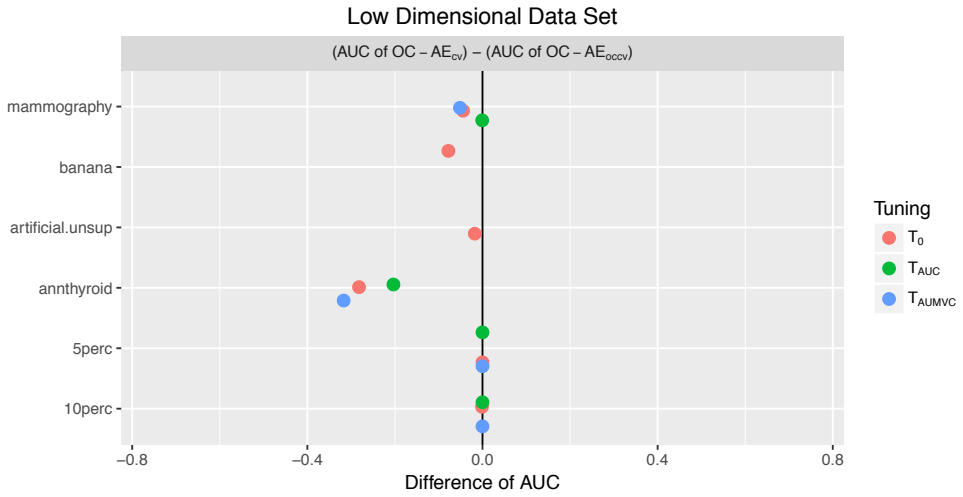


Figure 30: Visualisation of the difference between the AUC resulting from AE_{cv} and the AUC resulting from AE_{occv} (= vertical benchmark line) applied on low dimensional data sets. Examples on the right hand side of the benchmark line indicate that using AE_{cv} outperforms AE_{occv}. Ideally, all examples should be on the right or on the benchmark line. In that case the requirement violation of OC-SVM training on only one-class doesn't matter. Note that AE was not tuned for *banana* and *artifical.unsup*.

9.2.4.2 AUMVChd on High Dimensional Data

On the high dimensional data sets, T_0, T_{AUC} and $T_{AUMVChd}$ don't have an affect on the performance of the *perc5hd* and *perc10hd* data sets, in all cases, but the AUC already yields high AUC (Figure 31). T_{AUC} with the AE_{occv} yields higher AUC than with OC-SVM_{occv}, but similar AUC to LOF_{cv} and OC-SVM_{cv}. As for the rest of the data, tuning was not possible, but the AUC with an arbitrarily chosen number of hidden units outperforms both OC-SVM on the penglobal and letter data sets.

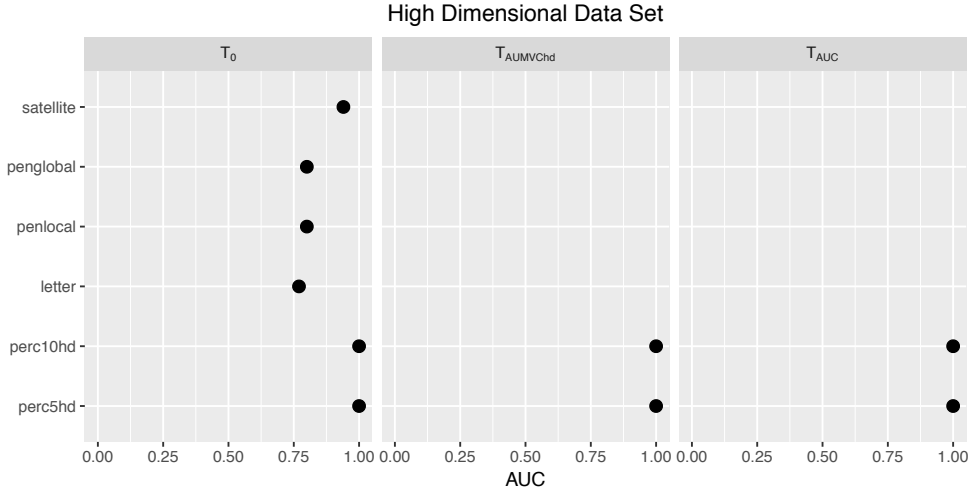


Figure 31: Aggregated AUC performance of the nested resampling results on high dimensional data sets generated by an autoencoder, when using T_0, $T_{AUMVChd}$ and T_{AUC}. The figure is grouped by tuning type, the left panel shows the results for T_0, the middle panel for $T_{AUMVChd}$ and the right panel shows the result for T_{AUC}.

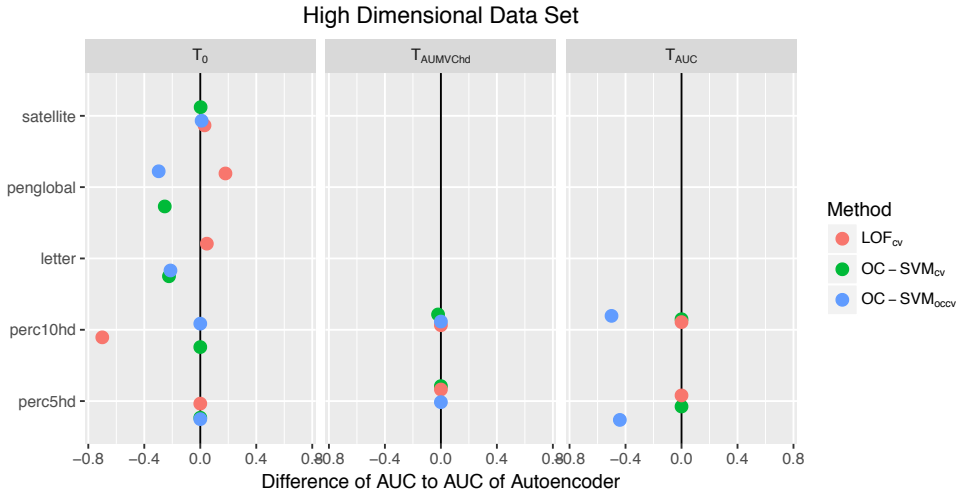


Figure 32: Visualisation of the difference between the AUC resulting from OC-SVM_{cv}, OC-SVM_{occv} LOF_{cv} and the AUC resulting from AE_{occv} (= vertical benchmark line), applied on high dimensional data sets. The figure is grouped by tuning type, the left panel shows the results for T_0, the middle panel for $T_{AUMVChd}$ and the right panel shows the result for T_{AUC}. Examples on the right-hand side of the benchmark line indicate that using AE_{occv} outperforms the other methods.

10 Summary

Anomaly detection is a special branch of machine learning and several anomaly detection methods exist. However, working in a semi-supervised or unsupervised settings is a challenge. This work covers the analysis of selected unsupervised and semi-supervised anomaly detection methods for point anomalies.
One of the challenges for successfully using anomaly detection methods is to evaluate and tune hyperparameters without labels, thus a new measurement method for low and high dimensional data called Area Under the Mass Volume Curve (AUMVC(hd)) is introduced (Section 3.2) and investigated (Section 9).
Among the methods used are the one-class support vector machines (OC-SVM) (Section 4.1), the local outlier factor (LOF) (Section 4.2) and the autoencoder (AE) (Section 4.3). OC-SVMs and autoencoders require training data containing only the normal class (semi-supervised), while LOF can work in the unsupervised setting. The OC-SVM learns a line, plane, hyperplane or sphere based on one class (the normal class) in order to separate all observations from that one class from the origin in the feature space. The assumption is, that when passing anomalous data into the learned model, the anomalous observations lie on the wrong side of that separation. The autoencoder detects anomalies by learning a model that is able to reconstruct the normal input data, after it compresses and decompresses them. The assumption is, that when passing anomalous data into the learned model, it can't be reconstructed like a normal observation, hence returning a high reconstruction error. The LOF uses the neighbours of each input point to determine if it is an anomalous observation by taking into account the distance and the local densities of its neighbours. If the observation has a sufficient lower density than its neighbors, it is considered as an anomaly.
The outputs of the introduced methods are score values that can have values on the complete range of the real numbers, to have a standardised output, a calibration function is applied (Section 6) to convert those scores into the interval $[0, 1]$.
As two of the used methods in this work are designed for one-class classification, additional resampling strategies are created to train only on one class in order to meet the methods' requirement when tuning and evaluating the respective methods (Section 5).
The introduced measurements, functions, methods and resampling strategies for anomaly detection were implemented in `mlR` and can be applied following the instructions in Section 7.4.
With these implementations, benchmark experiments were conducted on 6 low dimensional and 6 high dimensional data sets (Section 8.1 on data sets) with the OC-SVMs and LOF (Section 8). Due to computational challenges, the autoencoder was analysed separately (Section 9.2.4).
The performance of the AUMVC(hd) and the anomaly detection methods OC-SVM and LOF were investigated. The results show, that increasing the parameter values of the AUMVC(hd) (e.g. number of Monte-Carlo samples) leads to a longer runtime but not a better performance (Section 9.1.1). However, the AUMVC(hd) values highly fluctuate (Section 9.1.2), therefore more experiments and parameter values should be tested

in order to yield more robust results. It should be noted, that even by increasing the parameter values (e.g Monte-Carlo samples, number of α splits etc.), the performance should improve in theory, but a standard user might not be able to apply those settings due to computational constraints. With respect to the performance (based on the examples) and the computation time, the default setting for AUMVC(hd) in `mlR` are set according to Table 7. The default AUMVC needs up to 112 times longer than AUC to evaluate on low dimensional data; on high dimensional data the time increases by a factor up to 122021, but the time highly depends on the data size.
The runtime when using the default AUMVC for tuning (T_{AUMVC}) on low dimensional data takes half the time compared to using AUC for tuning (T_{AUC}). Although T_{AUMVC} outperforms T_{AUC} in a few examples, in most cases the results are similar to no tuning (T_0). Therefore, it can't be concluded that using T_{AUMVC} instead of T_0 can always be recommended, but if one uses T_{AUMVC}, it works best with the OC-SVM and resampling strategy OC-CV (Section 9.2.1).
$T_{AUMVChd}$ performs on high dimensional data better than T_{AUMVC} on low dimensional data when comparing both to T_{AUC}, but not in terms of runtime. $T_{AUMVChd}$ needs 25 times longer to tune than T_{AUC}. In the majority of the examples, using $T_{AUMVChd}$ is better than not tuning at all (T_0), and it also works best with OC-SVM with OC-CV (Section 9.2.1).
The requirement for OC-SVM is to train on one class only. When testing if the violation of the requirement affects the performance of OC-SVM, Section 9.2.2 indicates that when using T_{AUMVC} on low dimensional data the violation tends to not have a strong negative effect, which is in contrast to T_0. However, in high dimensional data, the violation of the requirement tends to have a slightly negative effect in both T_0 and $T_{AUMVChd}$ cases. Applying AE on R_{cv} instead of R_{occv}, and thus violating the training assumptions, does not have a strong effect on the method, based on the examples on low dimensional data sets.
For the sake of comparing both methods, OC-SVM (with CV and OCCV) was benchmarked against the LOF method (with CV) in Section 9.2.3. The results reveal that one should prefer both OC-SVM instead of the LOF method for low dimensional data sets. However, the LOF method should be preferred on high dimensional data (Section 9.2.3). But when comparing the AE_{occv} in Section 9.2.4, AE_{occv} performs always at least as good as the other methods, when using T_{AUC} or T_{AUMVC}, when using T_0 the OC-SVM_{occv} yields on average better results.
All in all, using AUMVC(hd) for tuning is in some cases useful, although it could not be determined what characteristic of the experiment made tuning with AUMVC(hd) successful in comparison to not tuning at all. Because of this, further experiments and investigations about the correlation of using AUMVC(hd) and the data and experiment setting should be made. Additionally, the number of experiments should be increased to make more certain conclusions, especially due to the fact that the method of AUMVC(hd) is highly based on randomness.

References

https://www.h2o.ai/. Accessed: 2018-01-20.

mlr tutorial. Accessed: 2018-01-20.

Elke Achtert, Hans-Peter Kriegel, and Arthur Zimek. Elki: a software system for evaluation of subspace clustering algorithms. In Scientific and statistical database management, pages 580–585. Springer, 2008.

Charu C Aggarwal. Probabilistic and statistical models for outlier detection. In Outlier Analysis, pages 35–64. Springer, 2017.

Thomas Albert. Learning hyperparameters for unsupervised anomaly detection. URL https://github.com/albertcthomas/anomaly_tuning.

Mennatallah Amer, Markus Goldstein, and Slim Abdennadher. Enhancing one-class support vector machines for unsupervised anomaly detection. In Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description, pages 8–15. ACM, 2013.

Bernd Bischl, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Erich Studerus, Giuseppe Casalicchio, and Zachary M. Jones. mlr: Machine learning in r. Journal of Machine Learning Research, 17(170):1–5, 2016. URL http://jmlr.org/papers/v17/15-066.html.

Bischl Bischl. Fortgeschrittene computerintensive methoden, 2016.

Loic Bontemps, James McDermott, Nhien-An Le-Khac, et al. Collective anomaly detection based on long short-term memory recurrent neural networks. In International Conference on Future Data and Security Engineering, pages 141–152. Springer, 2016.

Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. SIGMOD Rec., 29(2):93–104, May 2000. ISSN 0163-5808. doi: 10.1145/335191.335388. URL http://doi.acm.org/10.1145/335191.335388.

Guilherme O Campos, Arthur Zimek, Jörg Sander, Ricardo JGB Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E Houle. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. Data Mining and Knowledge Discovery, 30(4):891–927, 2016.

Arno Candel, Viraj Parmar, Erin LeDell, and Anisha Arora. Deep learning with h2o, 2015.

Arno Candel, Erin LeDell, Viraj Pamar, and Anisha Arora. Deep learning with h2o. H2O.ai, Inc, Published by H2O.ai, Inc, 4 2017. URL http://docs.h2o.ai/h2o/latest-stable/h2o-docs/booklets/DeepLearningBooklet.pdf.

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. ACM Comput. Surv., 41(3):15:1–15:58, July 2009. ISSN 0360-0300. doi: 10.1145/1541880.1541882. URL http://doi.acm.org/10.1145/1541880.1541882.

Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST), 2(3): 27, 2011.

Stéphan Clémençon and Jérémie Jakubowicz. Scoring anomalies: a m-estimation formulation. In Artificial Intelligence and Statistics, pages 659–667, 2013.

Anh Hoang Dau, Victor Ciesielski, and Andy Song. Anomaly detection using replicator neural networks trained on examples of one class. In SEAL, pages 311–322, 2014.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. Journal of the royal statistical society. Series B (methodological), pages 1–38, 1977.

M Elter, R Schulz-Wendtland, and T Wittenberg. The prediction of breast cancer biopsy outcomes using two cad approaches that both emphasize an intelligible decision process. Medical physics, 34(11):4164–4172, 2007.

Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. A geometric framework for unsupervised anomaly detection. In Applications of data mining in computer security, pages 77–101. Springer, 2002.

Ludwig Fahrmeir, Thomas Kneib, and Stefan Lang. Regression: modelle, methoden und anwendungen. Springer-Verlag, 2007.

Ludwig Fahrmeir, Christian Heumann, Rita Künstler, Iris Pigeot, and Gerhard Tutz. Statistik: Der Weg zur Datenanalyse. Springer-Verlag, 2016.

Tom Fawcett. An introduction to roc analysis. Pattern recognition letters, 27(8): 861–874, 2006.

Eibe Frank, Mark Hall, Geoffrey Holmes, Richard Kirkby, Bernhard Pfahringer, Ian H Witten, and Len Trigg. Weka-a machine learning workbench for data mining. In Data mining and knowledge discovery handbook, pages 1269–1277. Springer, 2009.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. The elements of statistical learning, volume 1. Springer series in statistics Springer, Berlin, 2001.

Jing Gao and Pang-Ning Tan. Converting output scores from outlier detection algorithms into probability estimates. In Data Mining, 2006. ICDM'06. Sixth International Conference on, pages 212–221. IEEE, 2006.

Vicente Garcıa, Ramón Alberto Mollineda, and José Salvador Sánchez. Index of balanced accuracy: A performance measure for skewed class distributions. 4th IbPRIA, pages 441–448, 2009.

Paul Glaister. 75.41 a" flat" function with some interesting properties and an application. The Mathematical Gazette, 75(474):438–440, 1991.

Nicolas Goix. How to evaluate the quality of unsupervised anomaly detection algorithms? arXiv preprint arXiv:1607.01152, 2016.

Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. PloS one, 11(4):e0152173, 2016.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. `http://www.deeplearningbook.org`.

Frank E Grubbs. Procedures for detecting outlying observations in samples. Technometrics, 11(1):1–21, 1969.

Michael Hahsler and Matthew Piekenbrock. dbscan: Density Based Clustering of Applications with Noise (DBSCAN) and Related Algorithms, 2017. URL `https://CRAN.R-project.org/package=dbscan`. R package version 1.1-1.

Michael Hahsler, Matthew Piekenbrock, and Derek Doran. dbscan: Fast density-based clustering with r.

Douglas M Hawkins. Identification of outliers, volume 11. Springer, 1980.

Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. Outlier detection using replicator neural networks. In DaWaK, volume 2454, pages 170–180. Springer, 2002.

Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. Artificial intelligence review, 22(2):85–126, 2004.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. An introduction to statistical learning, volume 6. Springer, 2013.

Michael I. Jordan. Lecture script: The kernel trick. Website. Available at: https://people.eecs.berkeley.edu/ jordan/courses/281B-spring04/lectures/lec3.pdf; access on 5.Mai 2017.

Alexandros Karatzoglou, Alex Smola, Kurt Hornik, and Achim Zeileis. kernlab – an S4 package for kernel methods in R. Journal of Statistical Software, 11(9):1–20, 2004. URL `http://www.jstatsoft.org/v11/i09/`.

Alexandros Karatzoglou, David Meyer, and Kurt Hornik. Support vector machines in r. 2005.

Miroslav Kubat, Stan Matwin, et al. Addressing the curse of imbalanced training sets: one-sided selection. In ICML, volume 97, pages 179–186. Nashville, USA, 1997.

Vipin Kumar. Parallel and distributed computing for cybersecurity. IEEE Distributed Systems Online, 6(10), 2005.

Olga Lyudchik. Outlier detection using autoencoders. Technical report, 2016.

David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien, 2015. URL `https://CRAN.R-project.org/package=e1071`. R package version 1.6-7.

Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In Proceedings of the 22nd international conference on Machine learning, pages 625–632. ACM, 2005.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. Journal of Machine Learning Research, 12(Oct):2825–2830, 2011.

John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. Advances in large margin classifiers, 10(3):61–74, 1999.

David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

Kevin L Priddy and Paul E Keller. Artificial neural networks: an introduction, volume 68. SPIE press, 2005.

Jonathan Root, Jing Qian, and Venkatesh Saligrama. Learning efficient anomaly detectors from k-nn graphs. In Artificial Intelligence and Statistics, pages 790–799, 2015.

Bernhard Scholkopf. The kernel trick for distances. Advances in neural information processing systems, pages 301–307, 2001.

Bernhard Schölkopf, Robert C Williamson, Alex J Smola, John Shawe-Taylor, and John C Platt. Support vector method for novelty detection. In Advances in neural information processing systems, pages 582–588, 2000.

Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. Neural computation, 13(7):1443–1471, 2001.

Michael R Smith and Tony Martinez. Improving classification accuracy by identifying and removing instances that should be misclassified. In Neural Networks (IJCNN), The 2011 International Joint Conference on, pages 2690–2697. IEEE, 2011.

David MJ Tax and Robert PW Duin. Support vector data description. Machine learning, 54(1):45–66, 2004.

Albert Thomas, Stephan Clemencon, Feuilard Vincent, and Alexandre Gramfort. Learning hyperparameters for unsupervised anomaly detection. Anomaly Detection Workshop, ICML 2016, 2016.

Christopher Tong and Vladimir Svetnik. Novelty detection in mass spectral data using support vector machine method. Computing Science and Statistics, 34, 2002.

L. Torgo. Data Mining with R, learning with case studies. Chapman and Hall/CRC, 2010. URL http://www.dcc.fc.up.pt/~ltorgo/DataMiningWithR.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. SIGKDD Explorations, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198. URL http://doi.acm.org/10.1145/2641190.2641198.

Yanchang Zhao. R and Data Mining: Examples and Case Studies. Academic Press, Elsevier, December 2012. ISBN 978-0-123-96963-7. URL http://www.rdatamining.com/docs/RDataMining-book.pdf.

Appendix

A Additional Measurement for Imbalanced Class Sizes

The standard unsupervised binary classification measurements (e.g. in Section 3.1) are not all suitable for anomaly detection. Some of the measurements don't take into account the unbalanced class sizes. Campos et al. [2016, p.900] introduce performance measurement designed for evaluating anomaly detection methods, that have at least labels in the test data set. There are a view more measurements for anomaly detecion with labels, in the following a view are explained in more detailed and implemented in mlR.

R-Precision /Top p-Accuracy The R-Precision (or also called top p-accuracy) measures the relative number of true predicted anomalies in the top p ranks of the test set in relation to the number of anomalies on the entire test set [Dau et al., 2014, p.316]. Ideally, $p \in \{1, 2, ..., |O|\}$.

$$\frac{|o \in O : rank(o) \leq p|}{|O|}, \tag{58}$$

Precision at p (P@p) The Precision at p measures the relative number of true predicted anomalies in the top p ranks of the test set in relation to the number of considered ranks p, in other words it measures the "proportion of correct results in the top p ranks" [Campos et al., 2016, p.900]. If $p = |O|$ the P@p results in the R-Precision measure. Ideally, $p \in \{1, 2, ..., |O|\}$

$$\frac{|o \in O : rank(o) \leq p|}{p} \tag{59}$$

Average Precision (AP) Both P@p and adjusted P@p are sensitive to the choice of p [Campos et al., 2016, p.901]. It is a hurdle to choose an appropriate p, where p ideally lies in $\{1, 2, ..., |O|\}$. The average precision tackle this issue by averaging over P@p with $p \in \{1, 2, ..., |O|\}$.

$$\frac{1}{|O|} \sum_{o \in O} P@rank(o) \tag{60}$$

Adjusted P@p and Adjusted AP Both P@p and AP depends on the magnitude of O, therefore a comparison between different test sets is not possible. An adjustment is necessary if one wants to compare different data sets, especially if the number of outliers differs. Campos et al. [2016, p.900] proposes to adjust P@p and the average precision for chances. The adjusted P@p is

$$\text{adjusted P@p} = \frac{P@p - \frac{|O|}{N}}{1 - \frac{|O|}{N}} \tag{61}$$

with the maximal value of P@p

$$\text{Maximum Index of P@p} = \begin{cases} |O|/p & p > |O| \\ 1 & p \leq |O| \end{cases} \tag{62}$$

If p is a large value, the maximum is always $|O|/p$ (note: not implemented in mlr, maximum index of P@p is always 1) [Campos et al., 2016, p.900]. Adjusted AP is analogous [Campos et al., 2016, p.901].

All observations can have the same score, in that case in all precision measurements described above "the ties are broken arbitrarily but consistently" [Campos et al., 2016, p.892] to evaluate the top p ranks.

B Additional Results Tables

Filename	Status
AMVhdWrapper	added
FailureModel	modified
Learner_properties	modified
Make_measure_AMV	added
Make_measure_AMVhd	added
Make_measure_WAC	added
Make_measure_precision	added
Measure	modified
OneClassTask	added
Prediction	modified
Prediction_operators	modified
RLearner	modified
RLearner_oneclass_h2oautoencoder	added
RLearner_oneclass_ksvm	added
RLearner_oneclass_lof	added
RLearner_oneclass_lofactor	added
RLearner_oneclass_svm	added
ResampleDesc	modified
ResampleInstance	modified
ResampleInstances	modified
Task	modified
TaskDesc	modified
Task_operators	modified
analyzeFeatSelResult	modified
calculateConfusionMatrix	modified

Table 27: Part 1: A list of files in the `R` folder of the mlR-repository [Bischl et al., 2016], which were modified or added in order to make anomaly detection possible.

Filename	Status
calculateROCMeasures	modified
checkLearnerBeforeTrain	modified
checkTunerParset	modified
convertScoresToProbability	added
data sets	modified
evalOptimizationState	modified
getOOBPreds	modified
helpers	modified
listLearners	modified
listMeasures	modified
makeLearner	modified
measures	modified
performance	modified
plotLearnerPrediction	modified
plotResiduals	modified
predictLearner	modified
resample	modified
setPredictThreshold	modified
setPredictType	modified
setThreshold	modified
tuneThreshold	modified
zzz	modified
makeData	modified
arg_lrncl	modified
arg_task_or_type	modified

Table 28: Part 2: A list of files in the `R` folder of the mlR-repository [Bischl et al., 2016], which were modified or added in order to make anomaly detection possible.

Filename	Status
run-learners-oneclass	added
run-oneclass	added
helper_helpers	modified
helper_objects	modified
test_base_AMVhdWrapper	added
test_base_Learner_properties	modified
test_base_checkData	modified
test_base_checkTaskLearner	modified
test_base_helpers	modified
test_base_listLearners	modified
test_base_measures	modified
test_base_oneclass	added
test_base_predict	modified
test_base_prediction_operators	modified
test_base_resample_OCbs	added
test_base_resample_OCcv	added
test_base_resample_OCholdout	added
test_base_resample_OCrepcv	added
test_base_resample_OCsubsample	added
test_base_resample_repcv	modified
test_learners_all_oneclass	added
test_oneclass_AMV_AMVhd	added
test_oneclass_h2oautoencoder	added
test_oneclass_ksvm	added
test_oneclass_lofactor	added
test_oneclass_lof	added
test_oneclass_measures	added
test_oneclass_svm	added

Table 29: A list of files in the test folder of the mlR-repository [Bischl et al., 2016], which were modified or added in order to test new functions and features of anomaly detection in mlR.

Unit: seconds

expr	min	lq	mean	median	uq	max	neval	cld
auc	0.000565636	0.0006613770	0.0007435064	0.000705655	0.0007982145	0.001232298	100	a
nsim10.aumvc	0.005782905	0.0062457425	0.0071431101	0.006748909	0.0074389245	0.030951816	100	bc
nsim100.aumvc	0.006922472	0.0073323475	0.0083592697	0.007792881	0.0085454865	0.026068680	100	c
nsim1000.aumvc	0.017439717	0.0195231015	0.0215279333	0.020622082	0.0220678795	0.046271516	100	de
nsim10000.aumvc	0.127243937	0.1388003340	0.1453349612	0.143761274	0.1495674335	0.204318274	100	g
nsim100000.aumvc	1.244238583	1.3871612610	1.4115948505	1.399674150	1.4377730960	1.528306340	100	i
nalpha10.aumvc	0.017502423	0.0186299710	0.0200859037	0.019933348	0.0211980860	0.025247998	100	d
nalpha50.aumvc	0.018092133	0.0191945105	0.0206467096	0.020454952	0.0217946790	0.026867530	100	de
nalpha100.aumvc	0.018850586	0.0198245205	0.0214249538	0.020852301	0.0225558015	0.036906149	100	de
dim2.1050.auc	0.000557192	0.0006359385	0.0008858215	0.000692564	0.0007574615	0.015772448	100	a
dim4.1050.auc	0.000585750	0.0006852295	0.0007794518	0.000736524	0.0008089615	0.001695889	100	a
dim6.1050.auc	0.000572076	0.0006687285	0.0007611426	0.000718191	0.0008386605	0.001265803	100	a
dim2.1050.aumvc	0.018354108	0.0192983160	0.0209248355	0.020588465	0.0220327035	0.040140148	100	de
dim4.1050.aumvc	0.024826551	0.0266936215	0.0281184424	0.027834675	0.0291374375	0.044638788	100	f
dim6.1050.aumvc	0.022733469	0.0236689955	0.0261377981	0.025155772	0.0267132645	0.044777242	100	ef
dim2.10500.auc	0.001034111	0.0011397765	0.0012820607	0.001208394	0.0012892765	0.004469903	100	ab
dim4.10500.auc	0.001031741	0.0011247340	0.0012814461	0.001184285	0.0013514965	0.003803400	100	ab
dim6.10500.auc	0.001056646	0.0011455410	0.0014685688	0.001245328	0.0013905770	0.017812604	100	ab
dim2.10500.aumvc	0.107192613	0.1453554190	0.1500208472	0.153689546	0.1586913835	0.213508895	100	gh
dim4.10500.aumvc	0.120389163	0.1403056380	0.1463572630	0.145336839	0.1510464870	0.223527094	100	g
dim6.10500.aumvc	0.133952850	0.1431572235	0.1533557139	0.150549000	0.1586168420	0.203294231	100	h

Table 30: Comparison of runtime (in seconds) for different settings for AUC and AUMVC, each setting was repeated 100 times. The labels in the first row state the value of the parameter, which is the focus of the experiment or deviates from the default setting.

Unit: seconds

expr	min	lq	mean	median	uq	max	neval	cld
auc	5.253570e-04	6.288105e-04	7.536364e-04	6.617155e-04	7.082590e-04	9.301221e-03	100	a
nsim10.aumvchd	2.484797e+00	2.546170e+00	2.604807e+00	2.580746e+00	2.620041e+00	3.194307e+00	100	d
nsim100.aumvchd	2.565250e+00	2.617181e+00	2.682467e+00	2.655599e+00	2.700640e+00	3.071277e+00	100	d
nsim1000.aumvchd	3.243893e+00	3.312502e+00	3.385017e+00	3.347363e+00	3.422793e+00	3.817267e+00	100	e
nsim10000.aumvchd	9.933833e+00	1.012443e+01	1.024170e+01	1.021186e+01	1.034380e+01	1.075331e+01	100	f
nsim100000.aumvchd	7.872801e+01	7.930366e+01	8.036221e+01	7.982995e+01	8.070559e+01	8.899456e+01	100	g
subfeat2.aumvchd	2.865377e+00	2.938589e+00	2.999589e+00	2.975314e+00	3.042085e+00	3.454030e+00	100	de
subfeat5.aumvchd	3.249580e+00	3.311625e+00	3.390335e+00	3.355042e+00	3.436636e+00	3.871092e+00	100	e
amviters5.aumvchd	9.623837e-01	9.898515e-01	1.025228e+00	1.003862e+00	1.027381e+00	1.356405e+00	100	b
amviters20.aumvchd	1.710389e+00	1.758746e+00	1.793197e+00	1.784284e+00	1.814379e+00	2.062022e+00	100	c
amviters50.aumvchd	3.240303e+00	3.313979e+00	3.389217e+00	3.364498e+00	3.428346e+00	3.807166e+00	100	e
n1050.auc	5.405830e-04	6.277450e-04	7.532782e-04	6.625160e-04	6.975785e-04	9.695450e-03	100	a
n1050.aumvchd	3.226762e+00	3.310394e+00	3.387239e+00	3.352233e+00	3.414467e+00	3.943727e+00	100	e
n10500.auc	9.932940e-04	1.077141e-03	1.112251e-03	1.113876e-03	1.145063e-03	1.258996e-03	100	a
n10500.aumvchd	1.302034e+02	1.329896e+02	1.357182e+02	1.346395e+02	1.379009e+02	1.492573e+02	100	h

Table 31: Comparison of runtime (in seconds) for different settings for AUC and AUMVChd, each setting was repeated 100 times. The labels in the first row state the value of the parameter, which is the focus of the experiment or deviates from the default setting.

OC-SVM - perc5 (n = 1050, d = 4)

Repetition 1

Tuning Measurements		Additional Measurements			Runtime
/		auc = 1	10 n.sim = 98.65	100 n.sim = 79.59	0.49 s
			1000 n.sim = 77.66	10000 n.sim = 78.87	
auc	1		10 n.sim = 123.54	100 n.sim = 124.60	3.38 min
10 n.sim	67.91	auc = 0.99	1000 n.sim = 120.13	10000 n.sim = 199.33	0.39 min
100 n.sim	68.07	auc = 0.84			0.40 min
1000 n.sim	55.31	auc = 1			0.58 min
10000 n.sim	77.19	auc = 1			2.35 min

Repetition 2

Tuning Measurements		Additional Measurements			Runtime
/		auc = 1	10 n.sim = 95.74	100 n.sim = 80.36	0.34 s
			1000 n.sim = 79.83	10000 n.sim = 80.89	
auc	1		10 n.sim = 124.30	100 n.sim = 113.03	3.63 min
10 n.sim	60.12	auc = 0.86	1000 n.sim = 112.37	10000 n.sim = 112.64	0.38 min
100 n.sim	56.56	auc = 0.94			0.38 min
1000 n.sim	77.41	auc = 0.75			0.54 min
10000 n.sim	77.71	auc = 0.95			2.23 min

Repetition 3

Tuning Measurements		Additional Measurements			Runtime
/		auc = 1	10 n.sim = 83.29	100 n.sim = 83.31	0.35 s
			1000 n.sim = 84.72	10000 n.sim = 84.51	
auc	1		10 n.sim = 74.4	100 n.sim = 84.20	3.56 min
10 n.sim	80	auc = 0.99	1000 n.sim = 86.88	10000 n.sim = 85.29	0.37 min
100 n.sim	92.67	auc = 0.99			0.36 min
1000 n.sim	65.5	auc = 0.97			0.55 min
10000 n.sim	66.12	auc = 0.99			2.34 min

OC-SVM - mammography data (n = 7595, d = 6)

Repetition 1

Tuning Measurements		Additional Measurements			Runtime
/		auc = 0.80	10 n.sim = 0	100 n.sim = 0	2.05 s
			1000 n.sim = 41.78	10000 n.sim = 26.45	
auc	0.87		10 n.sim = 254.45	100 n.sim = 149.69	22.77 min
10 n.sim	0	auc = 0.81	1000 n.sim = 313.18	10000 n.sim = 294.08	15.70 min
100 n.sim	0	auc = 0.78			17.05 min
1000 n.sim	7.41	auc = 0.75			18.26 min
10000 n.sim	7.41	auc = 0.74			29.22 min

Repetition 2

Tuning Measurements		Additional Measurements			Runtime
/		auc = 0.80	10 n.sim = 0	100 n.sim = 0	0.04 min
			1000 n.sim = 14.47	10000 n.sim = 31.83	
auc	0.86		10 n.sim = 0	100 n.sim = 559.77	23.18 min
10 n.sim	1771.45	auc = 0.80	1000 n.sim = 195.56	10000 n.sim = 321.37	19.85 min
100 n.sim	0	auc = 0.79			18.93 min
1000 n.sim	5.93	auc = 0.77			22.30 min
10000 n.sim	16.5	auc = 0.74			34.28 min

Repetition 3

Tuning Measurements		Additional Measurements			Runtime
/		auc = 0.81	10 n.sim = 0	100 n.sim = 5.50	0.04 min
			1000 n.sim = 16.34	10000 n.sim = 28.35	
auc	0.87		10 n.sim = 3604.60	100 n.sim = 128.78	24.08 min
10 n.sim	0	auc = 0.78	1000 n.sim = 369.72	10000 n.sim = 348.58	20.44 min
100 n.sim	0	auc = 0.76			18.91 min
1000 n.sim	0	auc = 0.77			23.82 min
10000 n.sim	4.58	auc = 0.74			36.41 min

OC-SVM - annthyroid data (n = 7062, d = 6)

Repetition 1

Tuning Measurements		Additional Measurements			Runtime
/		auc = 0.62	10 n.sim = 410.74	100 n.sim = 3.23	2.5 s
			1000 n.sim = 246.37	10000 n.sim = 304.28	
auc	0.82		10 n.sim = 0	100 n.sim = 440.95	22.89 min
10 n.sim	0	auc = 0.67	1000 n.sim = 65.76	10000 n.sim = 136.17	13.25 min
100 n.sim	549.05	auc = 0.64			12.11 min
1000 n.sim	26.31	auc = 0.67			16.89 min
10000 n.sim	110.92	auc = 0.73			34.10 min

Repetition 2

Tuning Measurements		Additional Measurements			Runtime
/		auc = 0.62	10 n.sim = 0	100 n.sim = 384.10	2.6 s
			1000 n.sim = 349.06	1000 n.sim = 268.17	
auc	0.81		10 n.sim = 22123.84	100 n.sim = 22123.84	24.96 min
10 n.sim	789.43	auc = 0.68	1000 n.sim = 22332.19	10000 n.sim = 22250.08	13.87 min
100 n.sim	0	auc = 0.59			14.14 min
1000 n.sim	44.6	auc = 0.74			17.45 min
10000 n.sim	91.27	auc = 0.69			33.69 min

Repetition 3

Tuning Measurements		Additional Measurements			Runtime
/		auc = 0.62	10 n.sim = 0	100 n.sim = 33.11	3.0 s
			1000 n.sim = 125.00	10000 n.sim = 270.98	
auc	0.81		10 n.sim = 0	100 n.sim = 0	20.22 min
10 n.sim	1141.01	auc = 0.65	1000 n.sim = 164.02	10000 n.sim = 121.05	14.48 min
100 n.sim	172.24	auc = 0.76			16.24 min
1000 n.sim	119.81	auc = 0.67			16.24 min
10000 n.sim	120.2	auc = 0.66			34.54 min

Table 32: Table of the performance results generated by SVM from the kernlab `R` package for every repetition. Each block contains the results from one data set (perc5, mammograpgy, annthyroid) and shows the runtime as well as the value of the tuning measurement and additional measurements. The abbreviation *10 n.iters* stands for AUMVC with 10 Monte-Carlo samples.

LOF - perc5 (n = 1050, d = 4)

Repetition 1

Tuning Measurements		Additional Measurements			Runtime
\	1	auc = 0.83	1000 n.sim = 417.51	10000 n.sim = 417.50	2 s
auc			1000 n.sim = 424.99	10000 n.sim = 425.02	4.45 min
1000 n.sim	426.14	auc = 1			2.57 min
10000 n.sim	429.44	auc = 1			16.70 min

Repetition 2

Tuning Measurements		Additional Measurements			Runtime
\	1	auc = 1	1000 n.sim = 437.92	10000 n.sim = 437.92	2 s
auc			1000 n.sim = 428.44	10000 n.sim = 428.45	4.12 min
1000 n.sim	430.4	auc = 1			2.81 min
10000 n.sim	439.11	auc = 1			16.68 min

Repetition 3

Tuning Measurements		Additional Measurements			Runtime
\	1	auc = 0.93	1000 n.sim = 455.05	10000 n.sim = 455.05	2 s
auc			1000 n.sim = 437.32	10000 n.sim = 437.35	4.11 min
1000 n.sim	421.45	auc = 1			2.56 min
10000 n.sim	436.92	auc = 1			16.20 min

LOF - mammography (n = 7595, d = 6)

Repetition 1

Tuning Measurements		Additional Measurements			Runtime
\	0.78	auc = 0.72	1000 n.sim = 91501.3	10000 n.sim = 91476.18	3 s
auc			1000 n.sim = 88398.45	10000 n.sim = 88401.69	12.2 min
1000 n.sim	73574.9	auc = 0.70			9.50 min
10000 n.sim	79569.4	auc = 0.69			24.31 min

Repetition 2

Tuning Measurements		Additional Measurements			Runtime
\	0.76	auc = 0.71	1000 n.sim = 112531.50	10000 n.sim = 112553.2	3 s
auc			1000 n.sim = 89885.42	10000 n.sim = 89885.42	12 min
1000 n.sim	80693.87	auc = 0.69			9.34 min
10000 n.sim	118904.4	auc = 0.67			23.90 min

Repetition 3

Tuning Measurements		Additional Measurements			Runtime
\	0.77	auc = 0.73	1000 n.sim = 93135.97	10000 n.sim = 93151.53	3 s
auc			1000 n.sim = 77937.67	10000 n.sim = 77937.67	11.95 min
1000 n.sim	96216.07	auc = 0.67			9.33 min
10000 n.sim	92860.25	auc = 0.67			23.95 min

LOF - annthyroid (n = 7062, d = 4) - lof

Repetition 1

Tuning Measurements		Additional Measurements			Runtime
\	0.74	auc = 0.74	1000 n.sim = 352274.11	10000 n.sim = 352274.11	3 s
auc			1000 n.sim = 330937.88	10000 n.sim = 330937.87	12.30 min
1000 n.sim	364164.91	auc = 0.74			9.21 min
10000 n.sim	352273.82	auc = 0.74			21.53 min

Repetition 2

Tuning Measurements		Additional Measurements			Runtime
\	0.73	auc = 0.73	1000 n.sim = 362408.64	10000 n.sim = 362408.64	3 s
auc			1000 n.sim = 348486.34	10000 n.sim = 348486.34	11.08 min
1000 n.sim	362408.64	auc = 0.74			8.42 min
10000 n.sim	301128.07	auc = 0.75			22.33 min

Repetition 3

Tuning Measurements		Additional Measurements			Runtime
\	0.73	auc = 0.73	1000 n.sim = 302424.89	1000 n.sim = 302440.50	3 s
auc			1000 n.sim = 371706.22	10000 n.sim = 371706.22	10.80 min
1000 n.sim	352564.7	auc = 0.75			8.44 min
10000 n.sim	310700.39	auc = 0.71			22.80 min

Table 33: Table of the performance results generated by LOF from the dbscan `R` package for every repetition. Each block contains the results from one data set (perc5, mammograpgy, annthyroid) and shows the runtime as well as the value of the tuning measurement and additional measurements. The abbreviation *10 n.iters* stands for AUMVC with 10 Monte-Carlo samples.

OC-SVM- perc5 (n = 1050, d = 4)

Repetition 1

Tuning Measurements		Additional Measurements			Runtime
	/				0.49 s
	auc = 1	50 n.alpha = 84.84	100 n.alpha = 83.72		
		150 n.alpha = 83.47			
auc	1	50 n.alpha = 124.49	100 n.alpha = 126.89		5.70 min
50 n.alpha	73.4	auc = 0.98	150 n.alpha = 125.31		
100 n.alpha	76.1	auc = 0.75			48.77 s
150 n.alpha	76.33	auc = 0.89			48.20 s
					47.4 s

Repetition 2

Tuning Measurements		Additional Measurements			Runtime
	/				0.49 s
	auc = 1	50 n.alpha = 78.32	100 n.alpha = 78.92		
		150 n.alpha = 77.95			
auc	0.9995	50 n.alpha = 80.58	amAlpha100 = 78.15		5.76 min
50 n.alpha	73.17	auc = 0.96	150 n.alpha = 77.89		43.14 s
100 n.alpha	79.35	auc = 0.81			45.81 s
150 n.alpha	74.25	auc = 0.91			53.79 s

Repetition 3

Tuning Measurements		Additional Measurements			Runtime
	/				0.47 s
	auc = 1	50 n.alpha = 85.69	100 n.alpha = 85.2		
		150 n.alpha = 83.73			
auc	1	50 n.alpha = 109.76	100 n.alpha = 107.59		6.15 min
50 n.alpha	66.97	auc = 0.98	150 n.alpha = 108.44		48.01 s
100 n.alpha	70.72	auc = 0.98			48.32 s
150 n.alpha	71.16	auc = 1			48.35 s

OC-SVM - mammography (n = 7595, d = 6)

Repetition 1

Tuning Measurements		Additional Measurements			Runtime
	/				2.74 s
	auc = 0.80	50 n.alpha = 29.20	100 n.alpha = 44.51		
		150 n.alpha = 13.73			
auc	0.87	50 n.alpha = 298.84	100 n.alpha = 300.24		26.24 min
50 n.alpha	0	auc = 0.64	150 n.alpha = 262.38		21.9 min
100 n.alpha	1	auc = 0.75			20.77 min
150 n.alpha	7.6	auc = 0.88			17.48 min

Repetition 2

Tuning Measurements		Additional Measurements			Runtime
	/				2.74 s
	auc = 0.80	50 n.alpha = 29.20	100 n.alpha = 44.51		
		150 n.alpha = 35.36			
auc	0.87	50 n.alpha = 292.29	100 n.alpha = 320.73		28.46 min
50 n.alpha	5.54	auc = 0.79	150 n.alpha = 233.02		20.25 min
100 n.alpha	15.54	auc = 0.79			20.89 min
150 n.alpha	0.12	auc = 0.67			18.73 min

Repetition 3

Tuning Measurements		Additional Measurements			Runtime
	/				2.74 s
	auc = 0.80	50 n.alpha = 62.49	100 n.alpha = 73.60		
		150 n.alpha = 22.13			
auc	0.87	50 n.alpha = 243.47	100 n.alpha = 354.85		30.15 min
50 n.alpha	9.04	auc = 0.75	150 n.alpha = 281.18		20.73 min
100 n.alpha	10.57	auc = 0.89			19.66 min
150 n.alpha	0	auc = 0.76			18.37 min

OC-SVM - annthyroid (n = 7062, d = 6)

Repetition 1

Tuning Measurements		Additional Measurements			Runtime
	/				1.98 s
	auc = 0.62	50 n.alpha = 441.29	100 n.alpha = 213.93		
		150 n.alpha = 381.79			
auc	0.81	50 n.alpha = 2447.91	100 n.alpha = 3288.91		18 min
50 n.alpha	108.65	auc = 0.74	150 n.alpha = 2471.21		17.86 min
100 n.alpha	264.88	auc = 0.73			17.42 min
150 n.alpha	209	auc = 0.58			15.78 min

Repetition 2

Tuning Measurements		Additional Measurements			Runtime
	/				1.9 s
	auc = 0.63	50 n.alpha = 552.25	100 n.alpha = 396.51		
		150 n.alpha = 150.43			
auc	0.82	50 n.alpha = 9012.25	100 n.alpha = 8376.61		20.31 min
50 n.alpha	100.26	auc = 0.72	150 n.alpha = 7237.51		16.63 min
100 n.alpha	155.34	auc = 0.67			17.05 min
150 n.alpha	71.48	auc = 0.67			16.31 min

Repetition 3

Tuning Measurements		Additional Measurements			Runtime
	/				1.6 s
	auc = 0.62	50 n.alpha = 253.75	100 n.alpha = 180.00		
		150 n.alpha = 125.78			
auc	0.81	50 n.alpha = 932.51	100 n.alpha = 3130.12		19.23 min
50 n.alpha	245.58	auc = 0.75	150 n.alpha = 1932.35		16.21 min
100 n.alpha	64.44	auc = 0.68			15.87 min
150 n.alpha	192.14	auc = 0.67			16.29 min

Table 34: Table of the performance results generated by SVM from the kernlab `R` package for every repetition. Each block contains the results from one data set (perc5, mammograpgy, annthyroid) and shows the runtime as well as the value of the tuning measurement and additional measurements. The abbreviation *50 n.alpha* stands for AUMVChd with splitting the alpha intervall into 50 subsamples.

Table 35: Table of the performance results generated by LOF from the dbscan `R` package for every repetition. Each block contains the results from one data set (perc5, mammograpgy, annthyroid) and shows the runtime as well as the value of the tuning measurement and additional measurements. The abbreviation *50 n.alpha* stands for AUMVChd with splitting the alpha intervall into 50 subsamples.

Table 36: Table of the performance results generated by SVM from the kernlab `R` package for every repetition. Each block contains the results from one data set (perc5hd, penlocal) and shows the runtime as well as the value of the tuning measurement and additional measurements. The abbreviation *5 amv.iters* stands for AUMVChd with 5 internal subsamples evaluated with AUMVC.

Table 37: Table of the performance results generated by LOF from the dbscan `R` package for every repetition. Each block contains the results from one data set (perc5hd, penlocal) and shows the runtime as well as the value of the tuning measurement and additional measurements. The abbreviation *5 amv.iters* stands for AUMVChd with 5 internal subsamples evaluated with AUMVC.

	Task	Learner	mean AUC	mean AUMVC	mean Runtime
1	perc5	oneclass.ksvm	1.00	79.62	0.30
2	perc5	oneclass.lof	0.94	446.78	0.62
3	perc10	oneclass.ksvm	1.00	51.79	0.16
4	perc10	oneclass.lof	0.64	115.77	0.61
5	annthyroid	oneclass.ksvm	0.62	408.33	2.46
6	annthyroid	oneclass.lof	0.73	368019.75	1.98
7	artificial.unsup	oneclass.ksvm	1.00	0.09	0.67
8	artificial.unsup	oneclass.lof	0.94	2.16	1.05
9	banana	oneclass.ksvm	0.62	1.15	1.20
10	banana	oneclass.lof	0.49	2.43	1.77
11	mammography	oneclass.ksvm	0.81	24.49	5.41
12	mammography	oneclass.lof	0.77	80663.16	3.56

Table 38: Benchmark results from the mlR package: Evaluation of the default SVM and default LOF method (no tuning) with AUC, AUMVC and runtime using 3-fold cross validation on 6 low dimensional data sets

	Task	Learner	mean AUC	mean AUMVC	mean Runtime
1	perc5	oneclass.ksvm	1.00	183.18	0.05
2	perc10	oneclass.ksvm	1.00	103.87	0.04
3	annthyroid	oneclass.ksvm	0.92	538.93	1.15
4	artificial.unsup	oneclass.ksvm	1.00	0.09	0.22
5	banana	oneclass.ksvm	0.92	1.53	0.20
6	mammography	oneclass.ksvm	0.85	25.13	3.16

Table 39: Benchmark results from the mlR package: Evaluation of the default SVM and default LOF method (no tuning) with AUC, AUMVC and runtime using 3-fold one-class cross validation on 6 low dimensional data sets

	Task	Learner	mean AUC	mean AUMVC	mean Runtime
1	perc5	oneclass.ksvm.tuned	1.00	81.38	36.00
2	perc5	oneclass.lof.tuned	1.00	446.76	87.36
3	perc10	oneclass.ksvm.tuned	1.00	63.82	36.43
4	perc10	oneclass.lof.tuned	1.00	115.77	105.08
5	annthyroid	oneclass.ksvm.tuned	0.82	11356.97	680.38
6	annthyroid	oneclass.lof.tuned	0.74	368019.75	527.53
7	artificial.unsup	oneclass.ksvm.tuned	1.00	0.10	75.71
8	artificial.unsup	oneclass.lof.tuned	1.00	2.16	214.29
9	banana	oneclass.ksvm.tuned	0.73	1.13	312.16
10	banana	oneclass.lof.tuned	0.67	2.44	349.68
11	mammography	oneclass.ksvm.tuned	0.89	274.83	3544.62
12	mammography	oneclass.lof.tuned	0.81	80652.21	3544.89

Table 40: Benchmark results from the mlR package: Nested resampling of 6 low dimensional data sets with the SVM and LOF method. The inner resampling loop uses the 3-fold cross validation strategy for tuning, with tuning measurement AUC. The outer resampling loop uses the 3-fold cross validation strategy to asses the performance (AUC, AUMVC and runtime) of the learner.

	Task	Learner	mean AUC	mean AUMVC	mean Runtime
1	perc5	oneclass.ksvm.tuned	0.98	71.83	27.34
2	perc5	oneclass.lof.tuned	1.00	446.64	192.95
3	perc10	oneclass.ksvm.tuned	0.98	36.78	31.69
4	perc10	oneclass.lof.tuned	0.55	115.71	196.87
5	annthyroid	oneclass.ksvm.tuned	0.79	49.16	447.64
6	annthyroid	oneclass.lof.tuned	0.72	368019.75	629.20
7	artificial.unsup	oneclass.ksvm.tuned	1.00	0.10	76.30
8	artificial.unsup	oneclass.lof.tuned	1.00	2.16	307.90
9	banana	oneclass.ksvm.tuned	0.66	1.12	226.54
10	banana	oneclass.lof.tuned	0.47	2.42	460.07
11	mammography	oneclass.ksvm.tuned	0.79	1.83	925.65
12	mammography	oneclass.lof.tuned	0.75	80633.25	727.80

Table 41: Benchmark results from the mlR package: Nested resampling of 6 low dimensional data sets with the SVM and LOF method. The inner resampling loop uses the 3-fold cross validation strategy for tuning, with tuning measurement AUMVC. The outer resampling loop uses the 3-fold cross validation strategy to asses the performance (AUC, AUMVC and runtime) of the learner.

	Task	Learner	mean AUC	mean AUMVC	mean Runtime
1	perc5	oneclass.ksvm.tuned	0.67	363.06	35.05
2	perc10	oneclass.ksvm.tuned	0.76	110.16	30.29
3	annthyroid	oneclass.ksvm.tuned	0.77	180597.28	424.01
4	artificial.unsup	oneclass.ksvm.tuned	0.75	0.54	16.83
5	banana	oneclass.ksvm.tuned	0.57	1.67	98.34
6	mammography	oneclass.ksvm.tuned	0.84	34703.48	406.96

Table 42: Benchmark results from the mlR package: Nested resampling of 6 low dimensional data sets with the SVM method. The inner resampling loop uses the 3-fold one-class cross validation strategy for tuning, with tuning measurement AUC. The outer resampling loop uses the 3-fold one-class cross validation strategy to asses the performance (AUC, AUMVC and runtime) of the learner.

	Task	Learner	mean AUC	mean AUMVC	mean Runtime
1	perc5	oneclass.ksvm.tuned	0.95	139.29	28.92
2	perc10	oneclass.ksvm.tuned	0.86	77.91	28.49
3	annthyroid	oneclass.ksvm.tuned	0.92	31063.86	496.49
4	artificial.unsup	oneclass.ksvm.tuned	1.00	0.11	96.58
5	banana	oneclass.ksvm.tuned	0.93	1.70	103.13
6	mammography	oneclass.ksvm.tuned	0.73	0.86	671.77

Table 43: Benchmark results from the mlR package: Nested resampling of 6 low dimensional data sets with the SVM method. The inner resampling loop uses the 3-fold one-class cross validation strategy for tuning, with tuning measurement AUMVC. The outer resampling loop uses the 3-fold one-class cross validation strategy to asses the performance (AUC, AUMVC and runtime) of the learner.

	Task	Learner	mean AUC	mean AUMVChd	mean Runtime
1	perc5hd	oneclass.ksvm	1.00	754.55	0.05
2	perc5hd	oneclass.lof	1.00	3152.80	0.20
3	perc10hd	oneclass.ksvm	1.00	325.84	0.05
4	perc10hd	oneclass.lof	0.30	730.47	0.20
5	letter	oneclass.ksvm	0.55	73.83	0.16
6	letter	oneclass.lof	0.82	332.76	0.53
7	penglobal	oneclass.ksvm	0.83	18.55	0.12
8	penglobal	oneclass.lof	0.82	42.76	0.16
9	penlocal	oneclass.ksvm	0.55	19.17	1.49
10	penlocal	oneclass.lof	0.98	44.91	4.17
11	satellite	oneclass.ksvm	0.94	20.20	1.55
12	satellite	oneclass.lof	0.97	621.15	1.37

Table 44: Benchmark results from the mlR package: Evaluation of the default SVM and default LOF method (no tuning) with AUC, AUMVChd and runtime using 3-fold cross validation on 6 high dimensional data sets

	Task	Learner	mean AUC	mean AUMVChd	mean Runtime
1	perc5hd	oneclass.ksvm	1.00	3153.93	0.05
2	perc10hd	oneclass.ksvm	1.00	704.16	0.05
3	letter	oneclass.ksvm	0.56	79.90	0.13
4	penglobal	oneclass.ksvm	0.99	30.26	0.04
5	penlocal	oneclass.ksvm	0.50	19.52	1.55
6	satellite	oneclass.ksvm	0.95	68.21	1.42

Table 45: Benchmark results from the mlR package: Evaluation of the default SVM and default LOF method (no tuning) with AUC, AUMVChd and runtime using 3-fold one-class cross validation on 6 high dimenional data sets

	Task	Learner	mean AUC	mean AUMVChd	mean Runtime
1	perc5hd	oneclass.ksvm.tuned	1.00	801.81	36.59
2	perc5hd	oneclass.lof.tuned	1.00	3202.57	82.51
3	perc10hd	oneclass.ksvm.tuned	1.00	329.16	35.06
4	perc10hd	oneclass.lof.tuned	1.00	708.97	93.54
5	letter	oneclass.ksvm.tuned	0.92	72.54	75.56
6	letter	oneclass.lof.tuned	0.85	381.09	114.70
7	penglobal	oneclass.ksvm.tuned	0.98	15.64	30.10
8	penglobal	oneclass.lof.tuned	0.84	42.29	67.88
9	penlocal	oneclass.ksvm.tuned	0.97	17.94	215.73
10	penlocal	oneclass.lof.tuned	0.97	39.88	395.22
11	satellite	oneclass.ksvm.tuned	0.96	19.01	240.11
12	satellite	oneclass.lof.tuned	0.97	680.58	326.80

Table 46: Benchmark results from the mlR package: Nested resampling of 6 high dimensional data sets with the SVM and LOF method. The inner resampling loop uses the 3-fold cross validation strategy for tuning, with tuning measurement AUC. The outer resampling loop uses the 3-fold cross validation strategy to asses the performance (AUC, AUMVChd and runtime) of the learner.

	Task	Learner	mean AUC	mean AUMVChd	mean Runtime
1	perc5hd	oneclass.ksvm.tuned	1.00	733.80	677.39
2	perc5hd	oneclass.lof.tuned	1.00	3211.22	3700.59
3	perc10hd	oneclass.ksvm.tuned	0.98	336.42	700.61
4	perc10hd	oneclass.lof.tuned	1.00	717.33	3736.68
5	letter	oneclass.ksvm.tuned	0.83	72.54	993.94
6	letter	oneclass.lof.tuned	0.70	349.43	4398.69
7	penglobal	oneclass.ksvm.tuned	0.81	15.64	586.80
8	penglobal	oneclass.lof.tuned	0.87	41.52	3277.78
9	penlocal	oneclass.ksvm.tuned	0.94	17.78	7174.56
10	penlocal	oneclass.lof.tuned	0.97	40.73	13089.99
11	satellite	oneclass.ksvm.tuned	0.81	19.01	4899.06
12	satellite	oneclass.lof.tuned	0.96	680.58	5239.89

Table 47: Benchmark results from the mlR package: Nested resampling of 6 high dimensional data sets with the SVM and LOF method. The inner resampling loop uses the 3-fold cross validation strategy for tuning, with tuning measurement AUMVChd. The outer resampling loop uses the 3-fold cross validation strategy to asses the performance (AUC, AUMVChd and runtime) of the learner.

	Task	Learner	mean AUC	mean AUMVChd	mean Runtime
1	perc5hd	oneclass.ksvm.tuned	0.56	2962.69	36.25
2	perc10hd	oneclass.ksvm.tuned	0.50	692.90	36.12
3	letter	oneclass.ksvm.tuned	0.53	83.05	90.15
4	penglobal	oneclass.ksvm.tuned	0.97	30.36	37.51
5	penlocal	oneclass.ksvm.tuned	0.99	17.59	516.04
6	satellite	oneclass.ksvm.tuned	0.72	65.40	463.63

Table 48: Benchmark results from the mlR package: Nested resampling of 6 high dimensional data sets with the SVM method. The inner resampling loop uses the 3-fold one-class cross validation strategy for tuning, with tuning measurement AUC. The outer resampling loop uses the 3-fold one-class cross validation strategy to asses the performance (AUC, AUMVChd and runtime) of the learner.

	Task	Learner	mean AUC	mean AUMVChd	mean Runtime
1	perc5hd	oneclass.ksvm.tuned	1.00	3020.94	733.46
2	perc10hd	oneclass.ksvm.tuned	1.00	672.63	737.98
3	letter	oneclass.ksvm.tuned	0.92	78.80	1083.95
4	penglobal	oneclass.ksvm.tuned	0.98	29.71	660.40
5	penlocal	oneclass.ksvm.tuned	0.96	19.11	8059.58
6	satellite	oneclass.ksvm.tuned	0.97	64.84	6506.45

Table 49: Benchmark results from the mlR package: Nested resampling of 6 high dimensional data sets with the SVM method. The inner resampling loop uses the 3-fold one-class cross validation strategy for tuning, with tuning measurement AUMVChd. The outer resampling loop uses the 3-fold one-class cross validation strategy to asses the performance (AUC, AUMVChd and runtime) of the learner.

	Task	Learner	mean AUC	mean AUMVC	Tuning Measure	
1	perc5	oneclass.ae	1.00	87.29	OCCV	no Tuning
2	perc10	oneclass.ae	1.00	66.88	OCCV	no Tuning
3	annthyroid	oneclass.ae	0.93	2120.14	OCCV	no Tuning
4	mammography	oneclass.ae	0.81	257.30	OCCV	no Tuning
5	banana	oneclass.ae	0.58	1.55	OCCV	no Tuning
6	artificial.unsup	oneclass.ae	0.74	1.49	OCCV	no Tuning
7	perc5	oneclass.ae	1.00	89.94	OCCV	AUC
8	perc10	oneclass.ae	1.00	66.91	OCCV	AUC
9	annthyroid	oneclass.ae	0.85	2426.36	OCCV	AUC
10	mammography	oneclass.ae	0.86	924074.37	OCCV	AUC
11	perc5	oneclass.ae	1.00	92.57	OCCV	AUMVC
12	perc10	oneclass.ae	1.00	92.57	OCCV	AUMVC
13	annthyroid	oneclass.ae	0.98	2784.11	OCCV	AUMVC
14	mammography	oneclass.ae	0.82	322534.12	OCCV	AUMVC

Table 50: Results from the mlR package: Nested resampling of 6 low dimensional data sets with the autoencoder. The inner resampling loop uses the 3-fold one-class cross validation strategy for tuning, with T_0, T_{AUC}, T_{AUMVC}. The outer resampling loop uses the 3-fold one-class cross validation strategy to asses the performance (AUC, AUMVC) of the learner.

	Task	Learner	mean AUC	mean AUMVChd	Tuning Measure	
1	perc5hd	oneclass.ae	1.00	2100.36	OCCV	no Tuning
2	perc10hd	oneclass.ae	1.00	618.92	OCCV	no Tuning
3	satellite	oneclass.ae	0.94	160.37	OCCV	o Tuning
4	penlocal	oneclass.ae	0.80	28.81	OCCV	no Tuning
5	res_penglobal	oneclass.ae	0.80	28.81	OCCV	no Tuning
6	letter	oneclass.ae	0.77	93.16	OCCV	no Tuning
7	perc5hd	oneclass.ae	1.00	2886.55	OCCV	AUC
8	perc10hd	oneclass.ae	1.00	683.22	OCCV	AUC
9	perc5hd	oneclass.ae	1.00	2970.68	OCCV	AUMVChd
10	perc10hd	oneclass.ae	1.00	690.97	OCCV	AUMVChd

Table 51: Results from the mlR package: Nested resampling of 6 high dimensional data sets with the autoencoder. The inner resampling loop uses the 3-fold one-class cross validation strategy for tuning, with T_0, T_{AUC}, $T_{AUMVChd}$. The outer resampling loop uses the 3-fold one-class cross validation strategy to asses the performance (AUC, AUMVChd) of the learner. Note: The autoencoder could not tune on all data sets due to instability of the model.

	Task	Learner	mean AUC	mean AUMVChd	Tuning Measure	
1	5perc	oneclass.ae	1.00	76.77	CV	No Tuning
2	10perc	oneclass.ae	1.00	43.10	CV	No Tuning
3	annthyroid	oneclass.ae	0.65	1043.66	CV	No Tuning
4	mammography	oneclass.ae	0.77	499.09	CV	No Tuning
5	banana	oneclass.ae	0.50	1.63	CV	No Tuning
6	artificial.unsup	oneclass.ae	0.72	1.36	CV	No Tuning
7	5perc	oneclass.ae	1.00	82.84	CV	Tuning with AUC
8	10perc	oneclass.ae	1.00	50.38	CV	Tuning with AUC
9	annthyroid	oneclass.ae	0.64	529.04	CV	Tuning with AUC
10	mammography	oneclass.ae	0.86	145.12	CV	Tuning with AUC
11	5perc	oneclass.ae	1.00	81.73	CV	Tuning with AUMVC
12	10perc	oneclass.ae	1.00	43.40	CV	Tuning with AUMVC
13	annthyroid	oneclass.ae	0.67	1388.68	CV	Tuning with AUMVC
14	mammography	oneclass.ae	0.77	256.87	CV	Tuning with AUMVC

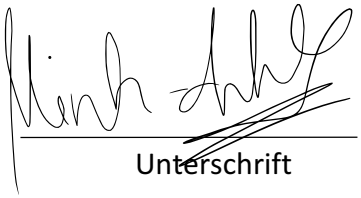
Table 52: Results from the mlR package: Nested resampling of 6 low dimensional data sets with the autoencoder. The inner resampling loop uses the 3-fold cross validation strategy for tuning, with T_0, T_{AUC}, T_{AUMVC}. The outer resampling loop uses the 3-fold cross validation strategy to asses the performance (AUC, AUMVC) of the learner. Note: The autoencoder could not tune on all data sets due to instability of the model.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die, aus fremden veröffentlichten oder nicht veröffentlichten Quellen, wörtlich oder sinngemäß übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

München, den 2.2.2018

Ort/Datum



Unterschrift