



Swiss Federal Institute of Technology Zurich

Seminar for
Statistics

Department of Mathematics

Master Thesis

Winter 2016

Mathias Mauchle

**A Comparison
of Mixed-Type
Anomaly Detection Methods**

Submission Date: May 7th 2017

Co-Adviser Dr. Manuel Proissl
Adviser: Prof. Dr. Meinshausen

To my lovely girlfriend Saskia

Acknowledgements

First and foremost, I would like to express my special appreciation and thanks to Prof. Dr. Meinshausen and the company Ernst & Young, in particular to Dr. Manuel Proissl. I want to thank Professor Meinshausen for his valuable guidance, helpful ideas and illuminating discussions during the development of my thesis and his support for the choice of the topic and the direction I developed the work.

I would like to thank Dr. Manuel Proissl for his helpful insights and his crucial advice for the direction of work and especially for the very helpful suggestions for the implementation part.

Further, I would like to express my sincere gratitude to Michael Strumpf who enabled me to get into contact with the company and without whom I would have never had this opportunity of writing an application-oriented master thesis in the first place.

In addition I want to thank my girlfriend Saskia for proofreading and sanity checking the content of my thesis.

Last but not least, I want to thank my parents and my girlfriend for supporting me mentally throughout writing this thesis and general support during my bachelor and master studies.

Abstract

This thesis describes and compares different anomaly detection methods, that are able to work with mixed-type data and are capable of dealing with a high number of data points. We motivate and define the necessary terms, used for anomaly detection and discuss different ideas for anomalies.

In addition, we describe the idea of different anomaly detection methods as well as their algorithms, which hyperparameters we need to choose for them and their advantages and disadvantages. Further we investigate how the methods handle different datatypes.

Furthermore we cover representation methods and combine them with anomaly detection methods, to build more powerful anomaly detectors.

In the experimental part we test how good the different anomaly detection methods work on real datasets and represented versions of them. Further we want to find out other properties of the methods such as robustness against noise and performance on different number of training samples. At the end we illustrate the scoring function of the anomaly detection methods on a two dimensional toy dataset.

Contents

Notation	xiii
1 Introduction	1
1.1 Context	1
1.2 Structure	3
1.3 Related Work	3
2 Important Definitions and Ideas	5
2.1 Input Data	5
2.2 Anomaly Type	6
2.3 Availability of Data Labels	6
2.4 Output of the Detection Algorithm	8
2.5 First Approach to Anomaly Detection	9
2.6 Novelty Detection in Detail	11
3 Anomaly Detection Methods	15
3.1 Isolation Forest	15
3.2 Unsupervised Random Forest and Related Methods	21
3.3 Cluster Based Methods	26
3.4 Autoencoder	34
3.5 Feature Regression and Classification	40
3.6 One-Class SVM	46
3.7 Least Squares Anomaly Detection	50
3.8 Probabilistic PCA	55
3.9 Gaussian Mixture Model	59
4 Representation Models	63
4.1 Principal Component Analysis	64
4.2 Autoencoder	67
4.3 Entity Embedding	70
5 Experiments	73
5.1 Datasets	73
5.2 Testing Pipeline	79
5.3 Parallelization	80
5.4 Learning Curves with Original Data	82
5.5 Compression Curves with PCA Represented Data	87
5.6 Compression Curves with AE Represented Data	93
5.7 Compression Curves with EE Represented Data	98
5.8 Robustness Curves	103
5.9 Visualization in 2D	108
6 Summary	115
6.1 Conclusions	115
6.2 Problems and Limitations	116
6.3 Future Work	117
6.4 Potential Applications	118

6.5 The Big Picture	119
Bibliography	121
A Implementation - Methods and Testing Program	123
A.1 Details - Anomaly Detection Methods	123
A.2 Details - Representation Methods	125
A.3 Test Function	126
A.4 Test Program	126
A.5 GUI	128

List of Figures

2.1	Plot - Two Gaussian Modes	9
2.2	Plot - Two Gaussian Modes with Density Contour Lines	9
2.3	Plot - Novelty Detection Example - Two Gaussian Modes	11
3.1	Plot - Butterfly Dataset	22
3.2	Plot - Butterfly Dataset - Original and Resampled	22
3.3	Plot - Clustered Butterfly Dataset	27
3.4	Plot - Clustered Butterfly Dataset with Voronoi Cells	32
3.5	Schema - Simple Autoencoder	34
3.6	Schema - Single Neural Network Unit	35
3.7	Schema - Multilayer Autoencoder	38
3.8	Plot - One-class SVM Boundary - Butterfly Dataset	46
4.1	Plot - PCA Representation Example	65
4.2	Plot - Autoencoder Representation Example	68
4.3	Schema - Entity Embedding	70
5.1	t-SNE Plot - Forest Cover Type Dataset	74
5.2	t-SNE Plot - Credit Card Fraud Dataset	75
5.3	t-SNE Plot - Financial Dataset	76
5.4	t-SNE Plot - Network Intrusion Dataset	77
5.5	t-SNE Plot - Higgs Challenge Dataset	78
5.6	Schema - Novelty Detection Pipeline	80
5.7	Screenshot - Example for Methods Fitting and Scoring Times	81
5.8	Learning Curve Plot - Forest Cover Type Data	82
5.9	Learning Curve Plot - Credit Card Fraud Data	83
5.10	Learning Curve Plot - Financial Data	84
5.11	Learning Curve Plot - Network Intrusion Data	85
5.12	Learning Curve Plot - Higgs Challenge Data	86
5.13	Compression Curve - Forest Cover Type Data	88
5.14	Compression Curve - Credit Card Fraud - PCA	89
5.15	Compression Curve - Financial Data - PCA	90
5.16	Compression Curve - Financial Data - PCA	91
5.17	Compression Curve - Higgs Challenge data - PCA	92
5.18	Compression Curve - Forest Cover Type Data - AE	93
5.19	Compression Curve - Credit Card Fraud Data - AE	94
5.20	Compression Curve - Financial Data - AE	95
5.21	Compression Curve - Network Intrusion Data - AE	96
5.22	Compression Curve - Higgs Challenge Data - AE	97
5.23	Compression Curve - Forest Cover Type Data - EE	98
5.24	Compression Curve - Credit Card Fraud Data - EE	99
5.25	Compression Curve - Financial Data - EE	100
5.26	Compression Curve - Network Intrusion Data - EE	101
5.27	Compression Curve - Higgs Challenge Data - EE	102
5.28	Noise Curve - Forest Cover Type Data	103
5.29	Noise Curve - Credit Card Fraud Data	104
5.30	Noise Curve - Credit Card Fraud Data	105
5.31	Noise Curve - Credit Card Fraud Data	106

5.32	Noise Curve - Credit Carf Fraud Data	107
5.33	2D contour plot - IF method - Butterfly Data	108
5.34	2D contour plot - URF method - Butterfly Data	109
5.35	2D contour plot - KMD method - Butterfly Data	109
5.36	2D contour plot - KMC method - Butterfly Data	110
5.37	2D contour plot - AE method - Butterfly Data	110
5.38	2D contour plot - FRaC method - Butterfly Data	111
5.39	2D contour plot - OSVM method - Butterfly Data	111
5.40	2D contour plot - LSAD method - Butterfly Data	112
5.41	2D contour plot - PCA method - Butterfly Data	112
5.42	2D contour plot - GMM method - Butterfly Data	113
A.1	Screenshot - Configuration File Example	126
A.2	Schema - Test Program Dependencies	127
A.3	Screenshot GUI - Mode Selection	128
A.4	Screenshot GUI - Data Selection	129
A.5	Screenshot GUI - Representation Selection	129
A.6	Screenshot GUI - Anomaly Detection Method Selection	130
A.7	Screenshot GUI - Run Section	130
A.8	Screenshot GUI - Output Window	131

List of Tables

5.1	Overview Properties - Datasets	79
-----	--	----

Notation

Conventions and Definitions

- Bold letter variables $\mathbf{x}, \boldsymbol{\mu}$ indicate vector valued variables.
- We denote a dataset matrix with X , which can be written in terms of data points/entries, $X = (\mathbf{x}_1, \dots, \mathbf{x}_n)^T$ or in terms of features $X = (\mathbf{x}^1, \dots, \mathbf{x}^p)$.
- $\delta_{i,j} := \begin{cases} 1, & i = j \\ 0, & \text{else} \end{cases}$, denotes the Kronecker-Delta.
- $\lfloor x \rfloor$ denotes the floor function, i.e. the function of x that returns the biggest integer that is smaller or equal to x .

Acronyms

- *AE*: Autoencoder
- *AUC*: Area under the curve (area under the "false positive rate vs. true positive rate" - curve)
- *CV*: Cross-validation
- *DBN*: Deep Belief Network
- *DAE*: Deep autoencoder
- *dummy encoding, 0-1-encoding, one-hot encoding*: All mean the same, for a categorical feature with c categories generate c features which have value 1 if an entry is in the corresponding category or zero otherwise.
- *FRaC*: Feature Regression and Classification model
- *IF*: Isolation Forest
- *KM*: k -means (clustering)
- *KMC*: k -means cluster size model
- *KMD*: k -means distance model
- *LSAD*: Least squares anomaly detection model
- *OSVM*: One-class SVM model

- *PCA*: Principal Component Analysis
- *PPCA*: Probabilistic Principal Component Analysis
- *RBM*: Restricted Boltzmann Machine
- *SV*: Support Vector
- *SVM*: Support Vector Machine

Chapter 1

Introduction

A lot of research has been done to solve classification tasks. A multitude of different algorithms were invented and tested to do classification, in all sorts of different areas and datasets. However, practical machine learning situations often ask to solve classification problems, where one class is strongly under-represented or even missing at all.

Consider the task of credit card fraud detection, where the goal is to find a classifier that divides transactions into fraud and non-fraud. Unfortunately, its often the case that the only data available are a huge amount of samples of non-fraud and a few known for fraud. Further, the situation requires to have some power in detecting new, potential fraudulent pattern, that do not match the distribution of the already observed data.

But a machine learning driven detection of novelties is certainly not limited to fraud detection at all. Another application could be the automated detection of "strange" objects in astronomy data that don't fit to any of the already classified object classes.

Classical classification methods are not or only poorly suited to solve this kind of problems, because they require a complete picture of all the patterns that should be classified. To solve this tasks, we need to come up with new approaches. These techniques build the field of anomaly detection.

This thesis covers methods that aim to solve Novelty Detection problems with potentially mixed type datasets of moderate sizes. Different methods for solving this problem are described and tested. Further we test how to combine the anomaly detection methods with representation preprocessing. This is interesting due to the fact, that often anomaly detector performances improves significantly by using feature representation before the detector is applied.

1.1 Context

The field *Anomaly Detection* attempts to find pattern in data, that do not match the "normal" behaviour. These patterns are referred in the following by the terms **anomaly** or **outlier**, whereas we refer to the patterns that match the expected structure of the data as **normal** or **nominal** points.

As mentioned before, anomalies are patterns in data that do not reflect the normal behaviour of the data. Reading this abstract description one can already expect, that the

precise definition of an anomaly is not always a straight forward task. The definition is very clear for some tasks and quite problem specific for other tasks. This thesis will use a very general definition for anomalies. Further it uses examples for testing, where the definition of the normal and the anomaly class is clear from the context.

Focus of the Thesis

Since the field of anomaly detection is very large, the thesis has to choose a focus. The focus of the thesis lies on large datasets, with a potentially large number of features (but still $n > p$, i.e. more data points than features) which can be numerical or categorical (or even ordinal). Nevertheless, we will stay as broad as possible and do not focus on a special area of application. Further we get an impression on how we could improve the performance of the anomaly detection algorithms, by applying representation learning as a preprocessing step. We want to test, if we can learn a representation from the data itself, that allows us to generate better features. These new features should facilitate the detection task for the anomaly detection methods.

Further we will concentrate on point anomalies (see section 2.2) and will not assume any temporal structure of the data points.

Problems to Solve

We need to solve a variety of different tasks, in order to get a good performing anomaly detection pipeline for mixed-type data with a moderate number of samples and features.

- *very different numbers of features*: We should be able to get good results independent of the number of features, they can range from a dozen to well into the hundreds or thousands. In particular we focus on the typical case of data sets, with more than ten and up to some hundred features.
- *different feature types*: The methods we investigate, need to work on mixed type data, i.e. datasets with numerical and categorical variables.
- *anomaly detection performance*: We want the methods to misclassify as few as possible anomalies as normal and vice versa. Since the trade-off of these two types of errors under different anomaly ratios is difficult and makes comparison difficult, we use the AUC as our measure for detection performance. This measure is independent of a special choice of a threshold, because it only depends on the order of the anomaly scores.
- *fast and parallelizable algorithms*: We want that the methods can be computed in a reasonable amount of time, for a number of features $n \approx 100\,000$. Even more importantly the computation of scores for new data points should be as fast as possible. This is important, since this time is finally needed by a working system for anomaly monitoring. In addition, the methods should be parallelizable to speed up the computation time of model fitting and scoring of new data, if necessary.
- *useful feature representations*: The representation of the data should be chosen, such that it matches the requirements of the detection methods and improves their detection performance.

1.2 Structure

In chapter 2, we begin by stating the important definitions and ideas of anomaly detection, such as Novelty and Outlier detection, the definition of an anomaly, the ways to quantify anomalies and a collection ideas to approach anomaly detection.

In chapter 3, we will then state a multitude of different anomaly detection methods, and cover details, such as their anomaly assumptions, the concrete algorithms to compute the models, the (hyper-)parameters involved as well as strengths and weaknesses.

Since some algorithms have problems in handling categorical data, we try to solve this problem by adding a representation step before applying the anomaly models. This means we encode our original data and use this encoded version of the data, as input of the anomaly detection methods. A detailed description of these representation models is given in chapter 4.

The Experiments which test the combinations of data representations and anomaly detection methods under different circumstances are performed in chapter 5.

Chapter 6 ends the thesis by summarizing the most important results, emphasizes the limitations of the work and states possible future research

1.3 Related Work

Many anomaly detection surveys have been done by different authors. The anomaly detection survey by [Chandola, Banerjee, and Kumar \(2009\)](#) provides an extended and structured overview over the existing methods and their application areas. In addition there is a review specifically examining novelty detection, by [Pimentel, Clifton, Clifton, and Tarassenko \(2014\)](#).

However, comparatively little research has been done in the area of mixed-type anomaly detection. Some of the proposed methods are anomaly detection by Feature Regression and Classification [Noto, Brodley, and Slonim \(2012\)](#), a student- t based approach by [Lu, Chen, Chen, and Lu \(2013\)](#), a method using a mixed-variate extension of RBM's by [Do, Tran, Phung, and Venkatesh \(2016\)](#) and its upgrade, where the Mv. RBM's are stacked into DBN's and varyingly deep DBN's are ensembled [Do, Tran, and Venkatesh \(2016\)](#).

Further we have not seen any attempts of combining anomaly detection algorithms with representation methods. An extensive survey of representation learning methods has been done by [Bengio, Courville, and Vincent \(2013\)](#).

Chapter 2

Important Definitions and Ideas

In this chapter we state the important definitions to capture the environment, in which we want to perform anomaly detection, e.g. the type of input data, the type of anomalies considered, the availability of labeled data and the format in which we report anomalies. It succeeds by explaining, how we approach anomaly detection, which intuitions are useful for anomaly detection and which ideas of anomalies are used in different anomaly detection methods. The chapter ends, by stating a mathematical definition for Novelty Detection. The discussion in this chapter follows vastly the anomaly detection survey done by [Chandola et al. \(2009\)](#).

2.1 Input Data

The input data of an anomaly detection method consists of *data points* (also called *points*, *data instances*, *events*, *samples*, *observations*), each consisting of a set of *features* (also called *attributes*, *variables*, *dimensions*).

Datasets can contain various different types of features:

- *continuous*. The feature values are real or integer valued.
- *categorical*. The feature values are in a set of categories. The special case with only two categories is called binary.
- *ordinal*. The feature values are given by a set of categories which have an ordering but no further meaning, eg. no notion of distance between the categories.
- *other*. There exist various other types of features like character, picture, graph, audio, video etc. They are typically used more infrequently and are often converted/encoded into numerical and categorical features.

The thesis covers the two feature types numerical and categorical. Further the ordinal variables can be handled too by all the methods, if they are interpreted as simple categorical features with the drawback, that the ordinal information usually is lost.

2.2 Anomaly Type

Before we start to approach the problem of anomaly detection, we have to be clear about how we define an anomaly e.g. what kind of anomalies we want to find. Thus, we first define the general types of anomalies one can distinguish.

- *Point Anomaly*. If a data point doesn't match the behaviour of the rest of the data set, we call it a point anomaly.
- *Contextual Anomaly* (or *conditional anomaly*). If a data point doesn't match the behaviour of the data which are in a specific context, we call it a contextual anomaly. A data point holds therefore the following two types of attributes:
 - *Contextual attributes*. Attributes, which describe the context (or neighbourhood) of a data point.
 - *Behavioural attributes*. Attributes, which describe the properties of a data point which are not related to the context of the data point.

Thus a data point can be an anomaly in one context and a normal point in different context. The most common types of context data are time-series data and spatial data.

- *Collective Anomalies*. If a group of related data points does not match the behaviour of the entire data set, we call them collective anomalies.

In the thesis we will focus exclusively on point anomalies.

2.3 Availability of Data Labels

The term labeled data means in this context the ability to know, if that a specific data point is defined as a normal point or as an anomaly. The availability of labeled data points is in reality often quite restricted due to:

- *Expensive labelling process*. It often requires an enormous amount of effort to label a big amount of data in an exhaustive way.
- *Required accuracy and representativeness*. Its hard to (often manually) label a data set in an accurate way, such that all types of different anomalies are represented. This problem is even worsened by the fact, that data labelling often has to be done manually.
- *Rareness of anomalies*. The problems of the two last points get exponentiated by the fact, that anomalies are often very rare and thus, a lot data points have to be investigated to find even one anomalous data point.
- *Dynamic anomaly behaviour*. Another big problem we have to deal with, is that typical anomaly behaviour is dynamic and changes over time. Thus, we might not have seen certain anomaly behaviour so far, that will arise in the future.

Depending on the availability of labeled data, we distinguish three major types of anomaly detection tasks:

- *Supervised anomaly detection*: If we assume that a detection technique has labeled data available in the training phase as well as in the test phase, we call it a supervised anomaly detection technique. Thus, it is basically a classification task and is usually solved in a similar manner, i.e. one builds a predictive classification model, which learns to distinguish the classes normal and anomaly. There are two major characteristics that arise in supervised anomaly detection, which make it a special case of classification. The data is typically strongly imbalanced (a lot fewer anomalies) and the labeled records of anomalies are often not representative for the complete anomaly class.
- *Semi-Supervised anomaly detection*: If we assume that a detection technique has only data labeled as normal available in the training phase and a mixture of normal and anomaly data available in the test set, we call it a semi-supervised anomaly detection technique. Since these algorithms do only require normal data, they can be used for a much bigger domain of application. A typical approach to this detection task is to build a model of the normal data in the training phase and to compare the test samples to this model in the test phase to identify the outliers.
- *Unsupervised Anomaly Detection*: If we assume that a decision technique has no or only unlabelled training data available and a mixture of normal and anomaly data for testing, we call it an unsupervised anomaly detection technique. In this case, we have to make assumptions about the anomalies, usually that normal instances outnumber anomalous ones, in order to get a useful algorithm. One can simply adapt semi-supervised anomaly detection algorithms to work in an unsupervised manner, if one uses a mixed training set instead of one with only normal data and builds in some robustness of the learnt model against outliers in the training set.

Unfortunately, the names semi- / un- / supervised anomaly detection¹ can be misunderstood. The name semi-supervised implies in an not anomaly related task typically a training set with labeled as well as unlabelled data is provided, whereas here it stands for a training set containing only data with one label.

Thus, we will use different names in this thesis, which are more convincing and less misleading. We will refer to the semi-supervised anomaly detection task as **Novelty Detection** and to the unsupervised anomaly detection task as **Outlier Detection**.

¹This commonly used terms stem from the review about anomaly detection by [Chandola et al. \(2009\)](#).

2.4 Output of the Detection Algorithm

The two common used ways to report anomalies are:

- *Scores*. In this case the algorithm outputs a number, which quantifies the degree to which it thinks, that a given data point is an outlier. This is usually a real number in $[0, 1]$, where small numbers indicate normal data points and big values indicate anomalies. This scores provides us essentially with a ranking of the data points from least to most anomalous one.

But a word of caution is needed here, since the scores are often not well calibrated, meaning that for example the scores of a data set range between 0.43 and 0.53. This makes the interpretation of a single score very difficult. To analyse the data, one can either choose a threshold and investigate the data points with scores bigger than the threshold or one can choose a certain fraction of points with the largest scores.

Thus it makes sense, when comparing the performance of these scores with the actual labels, to use the AUC measure. The AUC has the advantage that it only cares about the ranking of the scores and ignores their actual numbers.

- *Labels*. In this case the output consists of a label "normal" or "anomaly" for each test data point. If the output of the detection algorithm is a score, it can be easily transformed into a label, by choosing a threshold. Nevertheless, the inverse is not possible.

As mentioned above, scores are often hard to interpret, since they often lie on special intervals. A way to compare how "anomalous" a data point, is to compare its score with scores computed on the training data. So let $\mathbf{x}_1, \dots, \mathbf{x}_n$, be the training samples and $s(\mathbf{x})$ the scoring function, then we define the *(train) rank score* of a point \mathbf{x} as

$$s_r(\mathbf{x}) = \frac{\#\{i : s(\mathbf{x}_i) \leq s(\mathbf{x})\}}{n}. \quad (2.4.0.1)$$

This score is nothing else than the ratio of training scores, that are smaller than the score of the new sample. Thus, a score of 0.2 means that only 20% of the training samples are smaller than the new score, which would strongly suggest that the point is normal. In contrast to that, a score of 0.95 means that 95% of the training samples have smaller scores and thus, suggests that this point may well be an outlier.

2.5 First Approach to Anomaly Detection

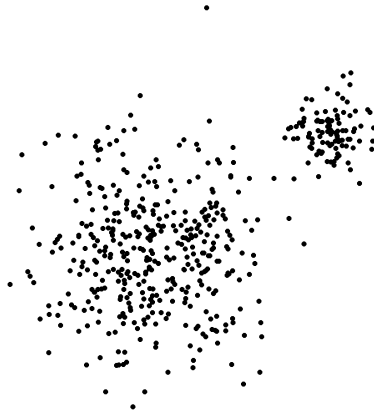


Figure 2.1: This figure shows a scatter plot of two Gaussian modes with different variances and one manually placed anomaly at the top.

As an introduction to anomaly detection, it helps to keep the picture of Figure 2.1 in our mind. A first simple anomaly definition is then to define points that lie far away from other ones as anomalies. We would intuitively think, that two clusters at the bottom-right and -left as normal data and the point at the top as an outlier or anomaly.

This idea can be formalized by the notion of density, i.e. we consider a point as an anomaly if it lies in a region of small density. Thus, we can perform density estimation (e.g. with kernel density estimation) which assigns different densities to different parts of the data space as we can see in Figure 2.2.

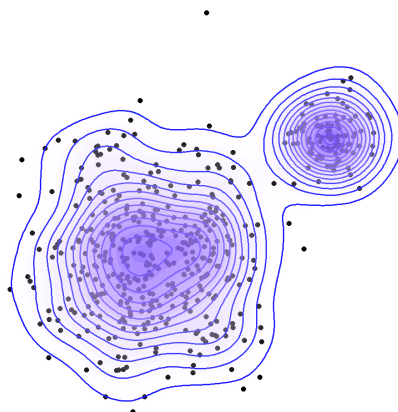


Figure 2.2: This figure shows the same scatter plot as in Figure 2.1 with a contour density estimate added to the scatter plot.

Although this toy example of Figure 2.1 is good for explaining the basic idea of anomaly detection, it doesn't appropriately represent the problems one faces in real data sets. In modern applications, the task consists of building anomaly models with a potentially high number of dimensions (not only two), mixed type data (numerical, categorical, ...) and a huge number data points (eg. in the millions). Unfortunately, the idea of traditional

density estimation can not be applied due to the *course of dimensionality* and the fact, that the density is not defined on mixed type data.

Another problem that arises when considering methods that involve distance, is the definition of the distance function. This choice can be challenging since it is not clear which distance metric definition between mixed type data points make sense. Additionally, there may be other aspects to define a point as an anomaly, which is not captured directly by the low-density definition. For example we could define data points that don't follow the same inter-feature relations like the normal data, as anomalies (see FRaC, see section 3.5).

Due to this problems we have to come up with new anomaly definitions and invent algorithms, that find data points fulfilling this definitions:

- *Isolation*: Consider a point as an anomaly, if they are easily isolated from all other points in the sense, that they can be isolated very fast by recursively splitting the data space along random features
- *High density regions wrt. reference data*: Define a point as an anomaly. if it lies in a low density region of the original data, compared to the density of of a new resampled reference set. The resampled set has asimilar marginal distribution, but no multivariate structure, i.e. it can be thought of as a "uniform background" compared to the original data.
- *Cluster distance/ inverse cluster size*: Assume that the data can be well represented by a certain number of clusters. Consider a point as an anomaly, if it doesn't match well the clustering structure. Two possible ways to define such a violation of the cluster structure are a cluster-center distance or a small cluster size.
- *Infrequent pattern*: Another idea is to learn a model, that represents the data in a compressed form (fewer features). Then we use big reconstruction errors as an indicator for anomalies, since we assume that the model learns to represent frequent pattern accurately but has a big reconstruction error, on infrequent pattern.
- *Depreciation by one-class classification*: The one-class classification idea is to capture as many data points as possible, using a spherical shaped boundary, while keeping the included area as small as possible. Points outside of the boundary are then classified as anomalies, where the degree of anomaly depends on the distance to the "middle" of the included volume.
- *Deviating feature relations*: The idea of feature modelling is to build for each feature a model given the other features. This models capture the common relations/distribution of the data and samples that deviate from this predictions, are considered anomalies.
- *Low density region of generative model*: Another interesting approach for an anomaly detection method is to assume, that the training data was produced by a generative model. Then one can fit the parameters of the generative model, compute a probability density over the data space and use small probabilities as a clue for anomalous samples.
- *Mixture model low-density regions* The idea of using the Gaussian mixture model is to model the density of the data and to consider points lying in low density regions as anomalies.

As mentioned before, the notion of density is not defined on mixed type data. Nevertheless, we could use this definitions of anomaly from before and the methods that are derived from them, as a tool to define and compute a density on mixed type data. As we will see in the next chapter about anomaly detection methods, the method to define how anomalous a data point \mathbf{x} is, will be the scoring function $s(\mathbf{x})$. This scoring function assigns to each point a value, that can be used for expressing the degree of anomaly.

Thus, a high number of training samples is generally expected in regions where $-s(\mathbf{x})$ is big. With this arguments, one can interpret $f(\mathbf{x}) = g(-s(\mathbf{x}))$ as a density function, where g is chosen as an appropriate monotone function. On the other hand, we can define for each density function f a scoring function $s(\mathbf{x}) = -h(f(\mathbf{x}))$, using an appropriate monotone function h . Therefore, there is a bilateral relation between anomaly detection methods and density estimation methods.

We will briefly mention of this relation for each method in the corresponding "Theoretical Modelling Approach"-section.

2.6 Novelty Detection in Detail

Novelty Detection is, as described in section 2.3, the task of building a model using training data of only one class and using the model in the second step to classify the test data that consists of the old class as well as a new class (not present in the training data).

The key issues when dealing with ND are nicely stated in the paper of [Markou and Singh \(2003\)](#) and we will summarize them in this section.

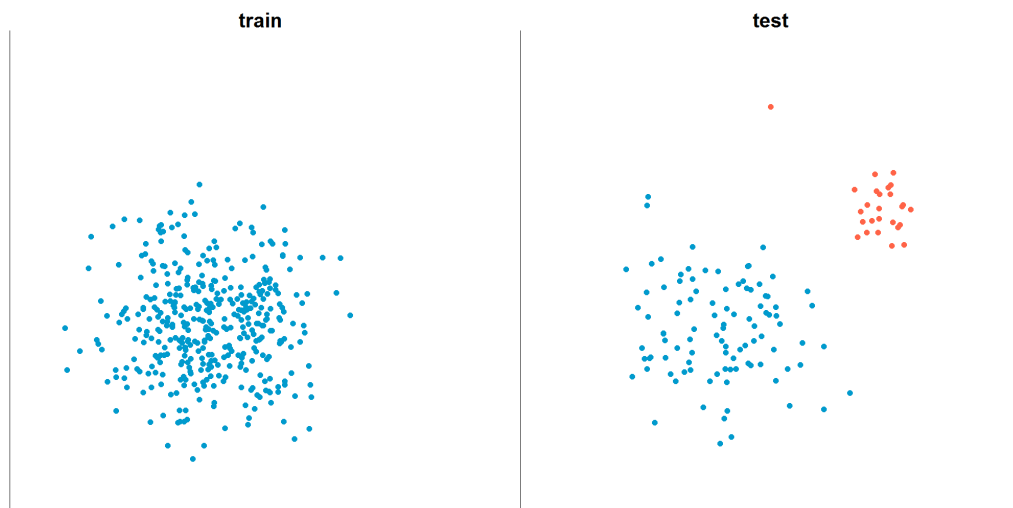


Figure 2.3: This scatter plot shows a simple example of a novelty detection scenario. The task is to build a model, that captures the distribution of the training data points at the left and classifies the test data points at the right into the classes "already seen" (normal) and "novel" (anomaly). A perfect anomaly detection method would give all red points bigger scores than the blue ones in the right picture.

Why Novelty Detection?

The methods of novelty detection are especially useful to detect new trends and patterns. For example, this can be used in an explorative sense to find new classes one didn't know yet. Often this points contain themselves particular valuable information, such as fraud cases in the scenario of transactional data or new trends in trading data.

Further, it is essential to a good classification or a system identification system to know, which object of the test data belong to the data patterns already known and which are new pattern, not known during the training phase.

Key Issues

As it is often the case in machine learning, there exists no best novelty detection algorithm (no free lunch theorems), i.e. that gives the best performance on every dataset. The usefulness of a particular ND algorithm in a ND system does strongly depend on the statistical property of the data. Key issues that arise in ND are:

- *Robustness vs. accuracy trade-off*: The ND algorithm should have a robust performance on the test set, where it maximizes the fraction of new patterns classified as novel data and minimizes the fraction of old patterns classified as new ones.
- *Minimal number parameters*: An anomaly detection algorithm should have as few free parameters as possible, which have to be chosen by the user. The reason is that they are quite difficult to choose. Especially if the algorithm performance highly depends on a specific choice of parameters which varies between data sets, the algorithm is highly susceptible to changing conditions (not robust) and difficult to be applied by a user.
- *Generalisation*: The algorithm should generalize well in the sense, that it doesn't mix up generalized normal data with anomalies.
- *Universality in input*: The algorithm should work independent of the number of features p , the fraction of anomalies (imbalance), low number of samples and noise features, i.e. features that don't hold any information about anomalies.
- *Computational complexity and parallelization*: Since ND algorithms often need to be refitted on a regular basis, they should have a computational complexity which is as small as possible. Further they should ideally be parallelizable, in order to use all the computational power available.

Formal Definition - Novelty Detection

Let $H_i, i = 1, \dots, m$ be independently identical distributed (iid) random variables (**assignment variable**) which are iid copies of a random variable H which satisfies

$$P(H = 0) = 1 - \epsilon, \quad (2.6.0.1)$$

$$P(H = 1) = \epsilon. \quad (2.6.0.2)$$

We define the **anomaly probability** as $\epsilon \in (0, 1)$ and the **anomaly ratio** as $r = \frac{\epsilon}{1-\epsilon}$.

Further let F, G be p -dimensional distribution functions. Then we define the **(Novelty) training distribution** as F and the **(Novelty) test distribution** as $F' = (1-\epsilon)F + \epsilon G$ or in more detail,

$$F' \sim \begin{cases} F, & \text{if } H_i = 0 \\ G, & \text{if } H_i = 1 \end{cases}. \quad (2.6.0.3)$$

Let $X_{train} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \sim F$ be the **(Novelty) training set** and $X_{test} = \{\mathbf{x}_1, \dots, \mathbf{x}_m\} \sim F'$ be the **(Novelty) test set**.

The **Novelty Detection Problem** then consists of constructing an estimator $\hat{H}(\mathbf{x}) \in [0, 1]$ of H only using X_{train} , such that the following expected value is as small as possible

$$E_{X_{test}} \left[L(\mathcal{H}, \hat{\mathcal{H}}(X_{test})) \right], \quad (2.6.0.4)$$

where

$$\mathcal{H} = (H_1, \dots, H_m)^T, \quad (2.6.0.5)$$

$$\hat{\mathcal{H}}(X_{test}) = \left(\hat{H}(\mathbf{x}_1), \dots, \hat{H}(\mathbf{x}_m) \right)^T. \quad (2.6.0.6)$$

In this thesis we choose as the loss function $L(\mathcal{X}, \mathcal{Y}) = 1 - AUC(\mathcal{X}, \mathcal{Y})$, where AUC is the area under the curve with \mathcal{X} the true assignment and \mathcal{Y} our estimation/scores.

Since the AUC function only depends on the ordering of \mathcal{Y} , it is invariant under monotone transformations of \mathcal{Y} and thus we don't need values that correspond to probabilities or even lie in the range $[0, 1]$. Therefore we can use any estimator $\hat{H}(\mathbf{x})$, that represents a monotone transformation of an estimator for the assignment variable H . In other words, the only thing the AUC measure considers, is the ordering of the scores, i.e. how near they come to the ideal case that the small scores all correspond to normal values and afterwards all bigger scores correspond to anomalies.

Chapter 3

Anomaly Detection Methods

In this chapter we present different anomaly detection methods, by stating their main idea, the anomaly assumptions and the concrete algorithm to compute them. Further we discuss how they handle different datatypes, how their hyperparameter are chosen and their dis-/advantages.

All algorithms, presented in this chapter, can in principle be used for Novelty and Outlier Detection but their usefulness varies considerably for the two scenarios. In Novelty detection the task generally consists of finding a model of the data, which is as good as possible whereas in Outlier Detection, the challenge is rather to find the right amount of restriction of the model complexity. The reason for that is to build in enough robustness into the model against the outliers, that are already present in the training set. In this thesis we will only focus on Novelty Detection though.

3.1 Isolation Forest

Whereas most anomaly detection approaches are based on modelling the "normal" class to detect anomalies, the Isolation Forest method ([Liu, Ting, and Zhou, 2008](#)) learns to isolate points from each other. The method finds anomalies by using an ensemble of decision trees specialized for isolating points.

Anomaly Assumption

The anomaly assumption in this anomaly detection method is that anomalies are "few and different", meaning that the anomalies

- i.) appear less often in the dataset than normal values and
- ii.) their feature values are substantially different from those of the normal samples.

The Isolation Forest method exploits this properties, by counting the number of recursive coordinate axis splits it takes, to isolate a sample from the rest of the dataset. We interpret this recursive random splits of the leftover samples as a decision tree. Thus the number of splits it takes to isolate a sample, corresponds to the path length from the root node to the external node in the tree.

Since the anomalies have the "few and different"-property they get isolated faster than normal data points. Thus they land in a leaf that is closer to the root compared to normal data points and thus, have shorter path lengths.

Algorithm

An Isolation Forest (iForest) consists, as the name suggests, of a forest of isolation trees (iTrees). The data sample is given by $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$. To build an isolation tree we pick in each step a feature $q \in \{1, \dots, p\}$ at random and split at a (uniformly distributed) random split point between *min* and *max* of the selected feature. This procedure gets repeated recursively until either

- i.) we reached the tree height limit l ,
- ii.) only one point is left in the set: $|X| = 1$,
- iii.) all points in X are identical.

Algorithm 1 $iTree(X, e, l)$

Inputs: X - input data, e - current tree height, l - height limit

Output: an iTree

```

1: if  $e \geq l$  or  $|X| \leq 1$  then
2:   return  $exNone\{Size \leftarrow |X|\}$ 
3: else
4:   let  $Q$  be a list of attributes in  $X$ 
5:   randomly select an attribute  $q \in Q$ 
6:   randomly select a split point  $s$  from max and min values of attribute  $q$  in  $X$ 
7:    $X_l \leftarrow filter(X, q < s)$ 
8:    $X_r \leftarrow filter(X, q \geq s)$ 
9:   return  $inNode\{$ 
10:      $Left \leftarrow iTree(X_l, e + 1, l),$ 
11:      $Right \leftarrow iTree(X_r, e + 1, l),$ 
12:      $SplitAtt \leftarrow q,$ 
13:      $SplitValue \leftarrow s\}$ 
14: end if
15: return iTree

```

We grow the isolation trees only until a height limit of $l = ceiling(\log_2(\psi))$ because this is approximately the average tree height. The reason for this bound on the tree height is that we only care about anomalies - which are expected to have path lengths which are shorter than the average.

Algorithm 2 $iForest(X, t, \psi)$ **Inputs:** X - input data, t - number of trees, ψ - subsampling size**Output:** a set of t $iTrees$

```

1: Initialize Forest
2: set height limit  $l = \text{ceiling}(\log_2 \psi)$ 
3: for  $i = 1$  to  $t$  do
4:    $X' \leftarrow \text{sample}(X, \psi)$ 
5:    $\text{Forest} \leftarrow \text{Forest} \cup iTree(X', 0, l)$ 
6: end for
7: return Forest

```

The first step to compute an anomaly score from an iForest is to compute the path length $h(\mathbf{x})$ for each iTree and then to compute the average path length $E[h(\mathbf{x})]$. Since the average path length is a positive real number we need to transform it to get an interpretable anomaly score.

Algorithm 3 $PathLength(\mathbf{x}, T, e)$ **Inputs:** \mathbf{x} - an instance, T - an iTree, e - current path length; to be initialized to zero when first called**Output:** path length of \mathbf{x}

```

1: if  $T$  is an external node then
2:   return  $e + c(T.size)$    $\{c(\cdot)$  is defined in Equation (3.1.0.1) $\}$ 
3: end if
4:  $a \leftarrow T.splitAtt$ 
5: if  $x^a < T.splitValue$  then
6:   return  $PathLength(\mathbf{x}, T.left, e + 1)$ 
7: else  $\{x^a \geq T.splitValue\}$ 
8:   return  $PathLength(\mathbf{x}, T.right, e + 1)$ 
9: end if

```

Its difficult to derive a score from $h(\mathbf{x})$ because the biggest possible tree height of an iTree grows with $\mathcal{O}(n)$ whereas the average height grows with order $\mathcal{O}(\log(n))$ and the normalization of $h(\mathbf{x})$ with any of these terms is either not bounded or cant be compared.

Using the interpretation of an iTree as a Binary Search Tree (BST) we can interpret the average path length $E[h(x)]$ as an unsuccessful search in BST. The average path length of an unsuccessful search in a BST with n samples is:

$$c(n) = 2H(n-1) - (2(n-1)/n), \quad (3.1.0.1)$$

where $H(i)$ is the harmonic number. We use $c(n)$ to normalize $h(\mathbf{x})$ and define the anomaly score as

$$s(\mathbf{x}) = 2^{-\frac{E[h(\mathbf{x})]}{c(n)}}, \quad (3.1.0.2)$$

where $E[h(\mathbf{x})]$ is the average path length of $h(\mathbf{x})$. If a sample \mathbf{x} gets classified in a terminal node with $Size > 1$ the path length e needs to be adjusted by $c(Size)$. The adjustment $c(Size)$ accounts for the average hight of a sub-tree beyond this node, which would have been built if the samples in the final node would have had slightly different values.

The anomaly score $s(\mathbf{x})$ behaves as:

- $s \rightarrow 0.5$, for $E[h(\mathbf{x})] \rightarrow c(n)$,
- $s \rightarrow 1$, for $E[h(\mathbf{x})] \rightarrow 0$,
- $s \rightarrow 0$, for $E[h(\mathbf{x})] \rightarrow 1$.

In addition $s(\mathbf{x})$ is monotonically increasing in $h(\mathbf{x})$. We can interpret the anomaly scores as follows:

- $s(\mathbf{x})$ close to 1: \mathbf{x} definitely an anomaly,
- $s(\mathbf{x}) \approx 0.5$: not sure if \mathbf{x} is an anomaly,
- $s(\mathbf{x})$ close to 0: \mathbf{x} definitely not an anomaly.

The usage of a sub-sampling size $\psi < 1$ is mainly important if we use the Isolation Forest as a Outlier Detection method. If we choose ψ close to one in an Outlier Detection scenario, this reduces the detection performance dramatically due to swamping and masking effects.

Swamping means that normal instances are wrongly classified as anomalies, which happens when normal instances are too close to anomalies. Thus the Isolation Forest needs more splits to separate anomaly samples which results in a bigger path length.

Masking refers to the effect that too many anomalies at the same place are hiding themselves as normal samples. If an anomaly cluster is too large and dense the Isolation Forest needs too many partitions to isolate these instances.

Another reason for subsampling is the rationale behind bagging ensemble methods, which says that each part of the ensemble sees some other part of the data and acts as an expert in the ensemble.

Theoretical Modelling Approach

A single isolation tree partitions the data space \mathbb{R}^p into axes parallel rectangles. Further the isolation forest, an ensemble of isolation trees, decomposes the space in smaller axes parallel rectangles. This is the case because the intersection of two axes parallel rectangles is either again an axis parallel rectangle or empty. The value of the scoring function $s(\mathbf{x})$, given by the average path length of all trees, is constant for all points in such a rectangle.

In more detail, let $\mathcal{P} = \{\mathcal{R}_1, \dots, \mathcal{R}_M\}$ be the partition of \mathbb{R}^p defined by the iForest. Then any two points $\mathbf{x}, \mathbf{x}' \in \mathcal{R}_r$ land in the same leaf in every iTree of the iForest and thus, have the same average path length: $E[h(\mathbf{x})] = E[h(\mathbf{x}')].$

Thus, we can write the estimator of the assignment function $H(\mathbf{x})$ the isolation forest as

$$\hat{H}(\mathbf{x}) = s(\mathbf{x}) = \sum_{r=1}^M \beta_r \mathbf{1}_{\{\mathbf{x} \in \mathcal{R}_r\}}, \quad (3.1.0.3)$$

where $\beta_r = h(\mathbf{x}) \in [0, 1]$, $\forall \mathbf{x} \in \mathcal{R}_r$ constant on each rectangle.

Therefore the isolation forest scoring function takes the form of a step function.

The IF is related to density estimation through the inverse path length, i.e. one could define a density through a monotone transformation of this quantity.

Handling Different Datatypes

The algorithm is originally designed for numerical features, but we can extend its usage to categorical and ordinal variables in a straight forward way.

- *categorical*: To do this we use the nice trick of lexical encoding, i.e. if we have a set of categories (*blue, red, black*) we just replace them with integers (1, 2, 3). Note that the order in which we enumerate the categories doesn't matter.
- *ordinal*: To encode ordinal variables we use the same trick of lexical encoding with the slight difference that we keep the natural order of the ordinal variables, i.e. (*small, medium, big*) is replaced by (1, 2, 3).

This encoding has the advantage that the feature space doesn't get blown up as it happens if we use dummy encoding.

Choice of Parameters

- **Sub-sampling size** $\psi \in (0, 1]$ (default: $\psi = 1.0$): Tests show that a subsampling size near or equal to 1.0 result in the best performance for the Novelty Detection scenario. This makes sense since for Novelty Detection it is important to build a model that represents the training data as accurate as possible.
- **Number of trees** $t \in \mathbb{N} \setminus \{0\}$ (default: $t = 100$): The performances of the method are stable against varying number of trees. Choosing the number of trees $t = 100$ often comes close to the optimum.

Advantages

- A strong advantage of the Isolation Forest is the robustness of the parameters, i.e. the performance of the algorithm is pretty stable for a wide range of the number of trees t . Further the performance doesn't change much if one uses values around the default choice of ψ .
- The method nicely integrates the categorical features using lexical encoding. This kind of encoding typically works well for trees and has the additional advantage, that the dimension of the feature space doesn't increase with encoding.
- The isolation forest algorithm has in the case of Outlier Detection has a very low memory requirement and a very low training as well as testing complexity (see (Liu et al., 2008)).
- The algorithm runtime does not increase with an increasing number of features. because building tree always requires the same amount of time, independent on the number of features.
- The algorithm is very simple to parallelize because the trees are built and evaluated independent of each other (in exactly the same way as supervised random forest method).
- The method is purely data driven, i.e. does not make any assumptions about the generative process of the data.

Disadvantages

- Using IF we often just achieve moderate performance with categorical data. The reason for this is the numerical nature of IF which is not thought for many equal values for one feature. Unfortunately that is what happens when we do lexical encoding, because all samples with the same category get the same value.

3.2 Unsupervised Random Forest and Related Methods

The Idea of this algorithm originates from the paper [Liaw and Wiener \(2002\)](#) and proposes to generate a synthetic anomaly dataset using the real data, whereon a Random Forest Classifier gets trained to distinguish those two datasets. The classifiers certainty for the synthetic dataset is then used as an anomaly score.

Anomaly Assumption

There are two different anomaly assumptions an unsupervised random forest (URF) can work with:

- *Low density region*: One assumption is that the real data lies in a region that has lower density than the synthetic generated data set. This is reasonable since the synthetic data forms something similar to a "uniform" distribution around and through the real data set. Thus the definition of a low density region compared to the real data is very similar to the definition of a region that has higher density with respect to the synthetic than with respect to the real data set.
- *Sampled anomaly set*: Another possible assumption is to presume, that the synthetic generated data set is a substitution of the anomaly dataset. This means we characterize an anomaly as a point that differs from the real data, by having a minimum fraction of dimensions that don't follow the structure of the normal data.

Algorithm

The first step of the URF algorithm, is to generate the synthetic dataset that represents the anomalies or uniform density, with which we want to compare the real data. The authors of the above mentioned paper propose a complete permutation of all values of each column, independent of each other. This complete permutation is reasonable if the number of features is relatively small (e.g. $p \approx 10$) but the produced data set is too different from the original data set in higher dimensions ($p \leq 50$). This is a consequence of the *course of dimensionality* and results in a classifier that fails to capture the boundaries of our data set narrow enough to detect anomalies.

To fix this shortfall, we only permute a fraction of each column and thus force our reference set samples not to have different values in each dimension, but only in a few. Of course the number of different values per sample can vary but is in expectation equal to the fraction α we permute per column, e.g. if we have $p = 100$ features and choose $\alpha = 0.2$, in expectation, approximately 20 feature values will be different from the original data point.

Algorithm 4 $shuffle(X, \alpha)$

Inputs: X - input data, α - fraction of each column to be resampled

Output: resampled data set X'

- 1: $X' = \text{copy}(X)$
 - 2: **for** *column* in X' **do**
 - 3: $\text{permute_fraction}(\text{column}, \alpha)$ // permute a fraction α of all entries of the column
 - 4: **return** X'
-

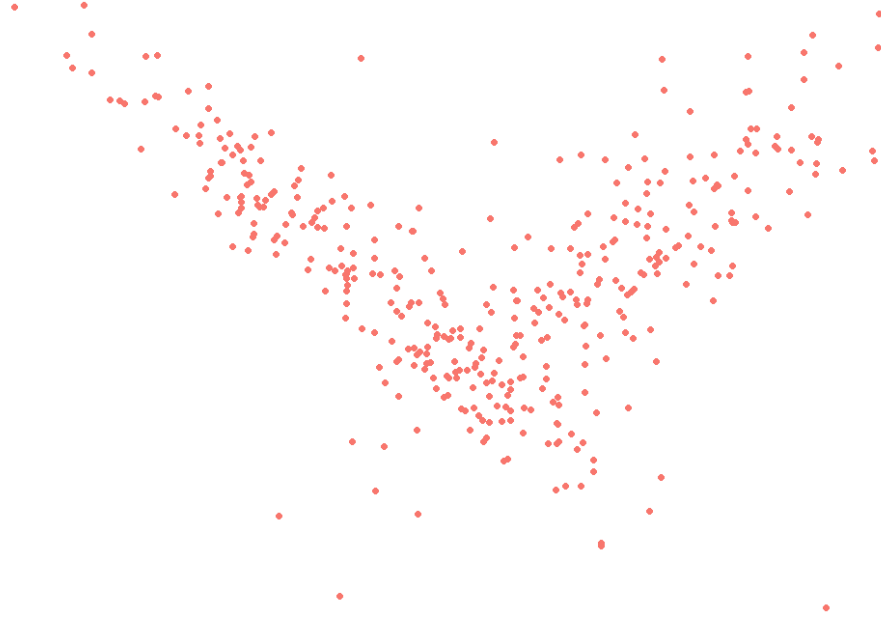


Figure 3.1: This scatter plot shows points sampled from two Gaussian modes with different covariance matrices. The dataset is used repeatedly through the thesis and we will refer to it as the "butterfly dataset", due to its shape.

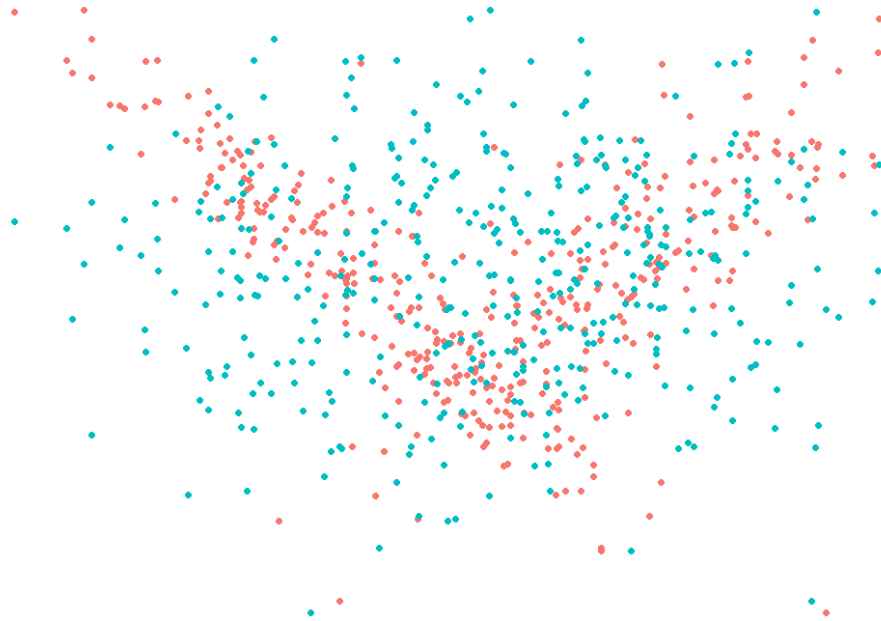


Figure 3.2: This figure shows the scatter plot of the butterfly dataset of Figure 3.1 and a resampled version of this data. The resampling was done by the *shuffle* function of Algorithm 4 with $\alpha = 1$.

The second step in the algorithm is to build a random forest classifier which learns to distinguish the original from the synthetic created reference dataset.

Algorithm 5 $URF(X, \alpha)$

Inputs: X - input data, α - fraction of each column to be resampled

Output: Random Forest Classifier (RFC)

- 1: Initialize RFC
 - 2: $\tilde{X} \leftarrow \text{shuffle}(X, \alpha)$
 - 3: $X' \leftarrow \text{concat}(X, \tilde{X})$, $label \leftarrow \text{create_label}(\text{length}(X), \text{length}(\tilde{X}))$
 - 4: $RFC \leftarrow \text{fit_RFClassifier}(X', label)$
 - 5: **return** RFC .
-

In principle we can replace the random forest classifier used by the algorithm by any other classification algorithm, e.g. by a tree boosting algorithm or a neural network. The reason why the random forest classifier works very well has different reasons:

- *few parameters:* The only parameter we have to choose for the random forest is the number of trees t , all other parameters we can leave at default. This is very helpful in the anomaly detection scenario, because we want to keep the number of parameters that need to be chosen manually, as small as possible.
- *robust parameters:* Another good property of the random forest parameters is that the number of trees is not critical to the performance of the classifier either. After growing a certain number of trees a random forest classifier keeps its performance even when quite a big number of additional trees are added. In contrast to this, other classifiers often have a big number of critical parameters to choose which makes it difficult to apply them. For example for boosted trees there are parameters such as the learning rate, the tree depth, subsampling rates for features and samples, etc. or in the case of neural networks there are the architecture of the network, number of training epochs, dropout rates, activation function, etc.
- *able to capture complicated borders:* Since the reference data set is quite similar to the real dataset and the "signal", that the anomaly data set holds, is distributed almost equally over all features. The classifier has to be able to capture very deep interactions and complicated boundaries which is ideal for a random forest with completely grown trees. This properties can well be observed when analysing the CV performance of the classifier. If we restrict the tree height or use another classifier instead, the performance significantly decreases.

Theoretical Modelling Approach

The unsupervised random forest method first creates a dataset, that should approximate the anomaly distribution $\tilde{X} \sim \tilde{G} \approx G$. Then we build a random forest classifier to distinguish \tilde{X} from the real dataset X . The decision function of a single tree of the random forest classifier is a step function, that defines a partition $\mathcal{P} = \{\mathcal{R}_1, \dots, \mathcal{R}_{M'}\}$ in the data space \mathbb{R}^p and is constant on the axes parallel rectangles \mathcal{R}_r . This decisions

function can be written as

$$g_{tree}(\mathbf{x}) = \sum_{r=1}^{M'} \beta_r \mathbf{1}_{\{\mathbf{x} \in \mathcal{R}_r\}}, \quad (3.2.0.1)$$

$$\beta_r = \frac{\sum_{i=1}^n Y_i \mathbf{1}_{\{\mathbf{x}_i \in \mathcal{R}_r\}}}{\sum_{i=1}^n \mathbf{1}_{\{\mathbf{x}_i \in \mathcal{R}_r\}}}, \quad (3.2.0.2)$$

where $Y_i \in \{0, 1\}$ is the label that specifies if a data point stems from the real 0 or the synthetic dataset 1 respectively. Further the decision function of a random forest classifier consists of the mean of the decision functions of single trees and defines again a partition of the space into axes parallel rectangles $\mathcal{P} = \{\mathcal{R}_1, \dots, \mathcal{R}_M\}$. The new axes parallel rectangles of the random forest that partition the space, are defined by the non-empty intersections of t rectangles, one from each tree in the forest. Thus the decision function of the forest, that we use as our decision function, takes the form of a step function

$$s(\mathbf{x}) := g_{forest}(\mathbf{x}) = \frac{1}{t} \sum_{j=1}^t g_{tree}^j(\mathbf{x}), \quad (3.2.0.3)$$

$$= \sum_{r=1}^M \beta_r \mathbf{1}_{\{\mathbf{x} \in \mathcal{R}_r\}}. \quad (3.2.0.4)$$

The URF method is related to density estimation through the local proportion of real data points. In other words, we can use a monotone transformation of $1 - \beta_r$ as a density estimate for a point $\mathbf{x} \in \mathcal{R}_r$.

Handling Different Datatypes

The encoding of categorical and ordinal variables works equal to the one of Isolation Forest. With the same advantages In addition its worth to mention that forests can handle lexical encoded categorical/ordinal variables very good.

Choice of Parameters

- **Random forest classifier parameters** (default $t = 200$):
The only parameter we have to chose for the random forest classifier are the number of trees t . Choosing the parameter $t = 200$ and fully grown trees performs well.
- **Fraction of each column to be resampled** $\alpha \in (0, 1]$ (default: $\alpha = 0.5$):
Choosing this equals 0.5 gives good results, but in general it should be adopted to the number of features. A possible way to do that would be to choose *alpha* such that fix the expected number of features that should be changed by the sampling mechanism. For example if we want to have in expectation 2 features that are different from the original sample with $p = 10$ features, we had to choose $\alpha = 0.2$.

Advantages

- The method can nicely deal with categorical variables using lexical encoding, without blowing up the data space with dummy encoding.

Disadvantages

- We have to store another dataset with the size of the training set.
- We have to choose the fraction of the columns that should be resampled. It is not clear a priori which fraction works the best, especially it potentially depends on quantities as the number of features and the ratio of categorical to numerical features.

3.3 Cluster Based Methods

Since clustering is one of most important unsupervised learning techniques to investigate data, its self-evident to investigate anomaly detection techniques based on clustering algorithms. We will focus on two different approaches to anomaly detection using k -means clustering, namely the cluster-center distance and the cluster size anomaly score.

Anomaly Assumption

The cluster distance approach uses the distance of a point to its assigned cluster center as an anomaly measure. This method is strongly linked to the distance approach to anomaly detection. For this algorithm we define datapoints as anomalies, if they lie far away from all others points.

The second approach uses the size of the cluster, to which a point is assigned, as the anomaly measure. This method is related to the idea of density approach to anomaly detection. The anomaly assumption for this algorithm is that points that lie in low density regions, are considered anomalous.

Nevertheless, both ideas are quite similar, because there is itself a close relationship between the "far away from other points"- and the "small density"-approach.

Algorithm

Our anomaly model consists of an ensemble of k -means (KM) clusterings with possible subsampling of columns and rows in each single clustering model. At first, we define the KM problem and its optimization algorithm. Note that we define the algorithms for the special case of a dataset with only numerical features. Later in the section "Handling Different Datatypes" it is described, how we can extend the complete algorithm to mixed data. We do this by only changing the distance measure, based on the euclidean norm, to a measure based on another norm, that can handle categorical data as well.

k -means problem *Given p -dimensional data points $X = (\mathbf{x}_1, \dots, \mathbf{x}_n)^T$, we try to find an assignment function $c : \mathcal{R}^d \rightarrow \{1, \dots, k\}$; $\mathbf{x} \mapsto c(\mathbf{x})$, which defines **prototypes** $\boldsymbol{\mu}_c \in \mathcal{Y} \subset \mathbb{R}^d$, $c \in \{1, \dots, k\}$ that minimize the following loss function*

$$L^{km}(c, \mathcal{Y}) = \sum_{\mathbf{x} \in X} \|\mathbf{x} - \boldsymbol{\mu}_{c(\mathbf{x})}\|^2 \quad . \quad (3.3.0.1)$$

The KM loss function (3.3.0.1) implies that the prototype vectors $\boldsymbol{\mu}_c$ are given by the mean of all data points that are assigned to the cluster c , which justifies the name of the clustering procedure, " k -means clustering". We will use the names *prototypes*, (*cluster*) *mean vectors* and (*cluster*) *centres/centroids* interchangeably and denote them by $\mathcal{Y} = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k\}$.

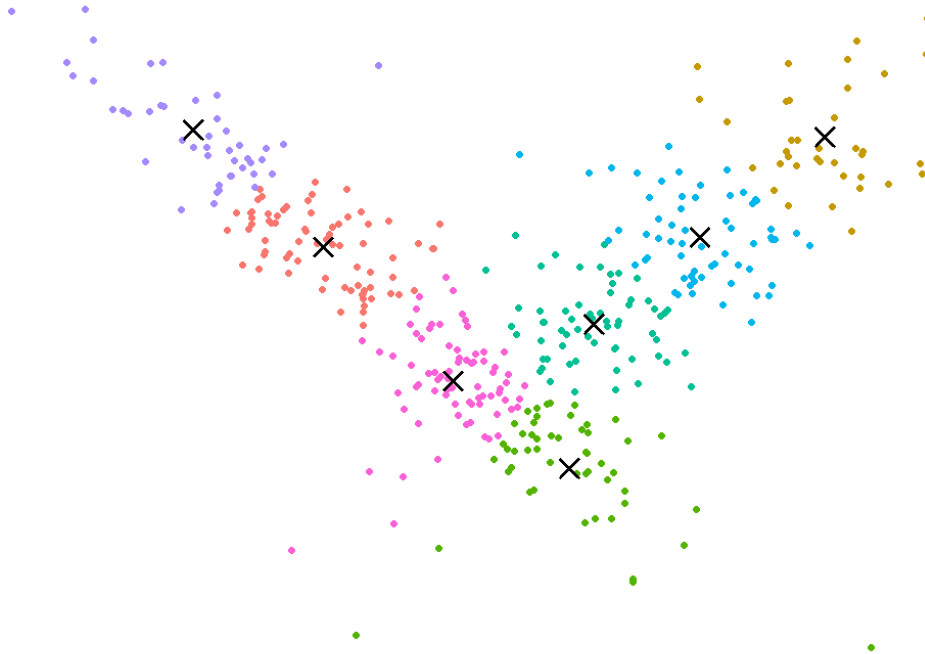


Figure 3.3: This figure shows the result of the k -means clustering algorithm on the butterfly dataset using $k = 7$ clusters.

The KM problem is a mixed combinatorial and continuous optimization problem, because the cluster assignment is combinatorial and the prototypes computation is continuous. Thus we can't solve it with a classical optimization method like gradient descent. Instead we use an algorithm especially designed for the KM problem, as shown in Algorithm 6.

In line 1 of the k -means algorithm (Algorithm 6), we initialize the starting cluster centres. This function "Initialize_prototypes(μ_1, \dots, μ_k)", that is used for initialization, can be crucial to find a good solution during the optimization process.

One can define several possible choices for the initialization of the k -means algorithm. These different procedures are displayed in Algorithms 7 - 9.

Algorithm 6 *k-means*(X, k)

Inputs: $X (= \{\mathbf{x}_1, \dots, \mathbf{x}_n\})$ - input data, k - number of clusters**Output:** set of k cluster mean vectors $\mathcal{Y} = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k\}$

- 1: $\{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k\} \leftarrow \text{Initialize_prototypes}(X)$
- 2: **repeat:**
- 3: Keep prototypes \mathcal{Y} fixed and assign sample vectors \mathbf{x} to nearest prototype

$$c(\mathbf{x}) = \arg \max_{c \in \{1, \dots, k\}} \|\mathbf{x} - \boldsymbol{\mu}_c\|^2 \quad (3.3.0.2)$$

- 4: Keep assignments $c(\mathbf{x})$ fixed and estimate prototypes

$$\boldsymbol{\mu}_\alpha = \frac{1}{n_\alpha} \sum_{\mathbf{x}: c(\mathbf{x}) = \alpha} \mathbf{x}, \quad \text{with} \quad n_\alpha = \#\{\mathbf{x} : c(\mathbf{x}) = \alpha\} \quad (3.3.0.3)$$

- 5: **until** changes of $c(\mathbf{x})$, \mathcal{Y} vanish
 - 6: $\mathcal{P} \leftarrow \text{cluster_properties}(X, \mathcal{Y})$ // save cluster properties like size and MSE
 - 7: **return** prototypes \mathcal{Y} , cluster properties \mathcal{P}
-

Algorithm 7 *Initialize_prototypes_random*(X, k)

- 1: **Inputs:** $X (= \{\mathbf{x}_1, \dots, \mathbf{x}_n\})$ - input data, k - number of clusters
 - 2: **Output:** set of k initial prototypes $\{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k\}$
 - 3: $\{i_1, \dots, i_k\} \leftarrow \text{sample_uniform}(\{1, \dots, n\}, \text{size}=k)$
 - 4: Define initial prototypes $\{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k\} \leftarrow \{\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_k}\}$
 - 5: **return** initial prototypes $\mathcal{Y} = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k\}$
-

Algorithm 8 *Initialize_prototypes_PlusPlus*(X, k)

- 1: **Inputs:** $X (= \{\mathbf{x}_1, \dots, \mathbf{x}_n\})$ - input data, k - number of clusters
- 2: **Output:** set of k initial prototypes $\{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k\}$
- 3: $i_1 \leftarrow \text{sample_uniform}(\{1, \dots, n\}, \text{size}=1)$
- 4: **for** $j \in 2, \dots, k$ **do:**
- 5: Calculate distances between $\boldsymbol{\mu}_{j-1}$ and all not yet selected points \mathbf{x}_i with

$$d(\boldsymbol{\mu}_{j-1}, \mathbf{x}_i) = \|\boldsymbol{\mu}_{j-1} - \mathbf{x}_i\|^2 \quad . \quad (3.3.0.4)$$

- 6: Define $P(i)$ as the probability function that specifies the probability of choosing \mathbf{x}_i as next prototype $\boldsymbol{\mu}_j$. We define $P(i)$ in terms of $d(\boldsymbol{\mu}_{j-1}, \mathbf{x}_i)$, s.t. the points with the biggest distances have higher probabilities of being chosen than such with smaller distances.
 - 7: Sample $i_j \sim P(i)$ // index of chosen sample in dataset X
 - 8: $\boldsymbol{\mu}_j \leftarrow \mathbf{x}_{i_j}$
 - 9: **return** initial prototypes $\mathcal{Y} = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k\}$
-

Algorithm 9 *Initialize_prototypes_furthest*(X, k)

Inputs: X ($= \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$) - input data, k - number of clusters**Output:** set of k initial prototypes $\{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k\}$

- 1: $i_1 \leftarrow \text{sample_uniform}(\{1, \dots, n\}, \text{size}=1)$
- 2: **for** $j \in 2, \dots, k$ **do**:
- 3: Calculate distances between $\boldsymbol{\mu}_{j-1}$ and all not yet selected points \mathbf{x}_i with

$$d(\boldsymbol{\mu}_{j-1}, \mathbf{x}_i) = \|\boldsymbol{\mu}_{j-1} - \mathbf{x}_i\|^2 \quad . \quad (3.3.0.5)$$

- 4: Choose the next prototype to be the point which is the furthest away from the last prototype and has not yet been selected:

$$\boldsymbol{\mu}_j = \arg \max_{x_i} d(\boldsymbol{\mu}_{j-1}, \mathbf{x}_i) \quad (3.3.0.6)$$

- 5: **return** initial prototypes $\mathcal{Y} = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k\}$
-

Since it can occur that the algorithm can converge very slowly or converge to non-optimal centroids, due to specific choices of initial points (see [Arthur and Vassilvitskii \(2007\)](#)), it is important to choose the right initialization procedure. It can be shown that the PlusPlus initialization, also known as k -means++ algorithm, has a lot of advantages compared to the other two initialization techniques we explained. Thus we choose the PlusPlus initialization for our k -means based anomaly detection method.

As mentioned before, we ensemble multiple KM models trained on only a subset of the features and a subset of all the available training data, to obtain more stable and better performing anomaly scores. This procedure of ensembling k -means models is displayed in the Algorithm 10.

Algorithm 10 *k-means ensemble*(X, n)

Inputs: X - input data, n - number of clusterings in the ensemble, ψ - subsample ratio, ϕ - colsample ratio**Output:** ensemble of n k -means clusterings $\mathcal{E} = \{\mathcal{Y}_1, \dots, \mathcal{Y}_n\}$

- 1: Initialize ensemble of k -means models $\mathcal{E} = \{\}$
 - 2: **for** $i = 1, \dots, n$ **do**
 - 3: $\tilde{X} \leftarrow \text{sample_rows_cols}(X, \phi, \psi)$ // sample fraction of rows and columns according to ratios
 - 4: $\mathcal{Y}_i \leftarrow k\text{-means}(\tilde{X}, k)$
 - 5: $\mathcal{E} \leftarrow \mathcal{E} \cup \mathcal{Y}_i$
 - 6: **return** \mathcal{E}
-

In a next step we define the algorithm that computes the anomaly scores for a single KM clustering model \mathcal{Y} . The algorithm for the calculation of the k -means distance (KMD) anomaly score is given by Algorithm 11.

Algorithm 11 $KMD_score(x, \mathcal{Y})$

Inputs: \mathbf{x} - new data point, \mathcal{Y} - KM prototypes**Output:** anomaly score $s(\mathbf{x})$

1: Compute distance to closest prototype

$$d(\mathbf{x}) = \min_{\boldsymbol{\mu} \in \mathcal{Y}} d(\mathbf{x}, \boldsymbol{\mu}) \quad (3.3.0.7)$$

2: Compute anomaly score

$$s(\mathbf{x}) = 1 - e^{-d(\mathbf{x})} \quad (3.3.0.8)$$

3: **return** $s(\mathbf{x})$

In a similar manner of the k -means distance anomaly score algorithm, we define the algorithm to compute the k -means clustersize (KMC) anomaly score, as written in Algorithm 12.

Algorithm 12 $KMC_score(x, \mathcal{Y})$

Inputs: \mathbf{x} - new data point, \mathcal{Y} - KM prototypes**Output:** anomaly score $s(\mathbf{x})$

1: Compute index of closest prototype

$$i(\mathbf{x}) = \min_{i \in 1, \dots, k} d(\mathbf{x}, \boldsymbol{\mu}_i) \quad (3.3.0.9)$$

2: Retrieve cluster size of closest cluster

$$cs(\mathbf{x}) = \mathcal{P}.cluster_sizes(i(\mathbf{x})) \quad (3.3.0.10)$$

3: Compute anomaly score

$$s(\mathbf{x}) = 1 - e^{-1/cs(\mathbf{x})} \quad (3.3.0.11)$$

4: **return** $s(\mathbf{x})$

Form the algorithms 11 and 12, we know how to calculate the KMD and the KMC anomaly scores of a single KM model. However, we still have to combine these different scores, which is described by Algorithm 13.

Algorithm 13 $ensemble_score(x, \mathcal{E})$

Inputs: \mathbf{x} - new data point, \mathcal{E} - KM ensemble**Output:** anomaly score $s(\mathbf{x})$ 1: **for** $\mathcal{Y}_i \in \mathcal{E}$ **do**:2: $s_i(\mathbf{x}) = KMD_score(x, \mathcal{Y}_i)$ // or $KMC_score(x, \mathcal{Y}_i)$ respectively3: **return** $s(\mathbf{x}) = combine_scores(s_1(\mathbf{x}), \dots, s_n(\mathbf{x}))$

The "combine_scores" function can be one of min, mean, median, max. Experiments show, that the max often performs best for the novelty detection task.

Theoretical Modelling Approach

The scoring functions of both k -means clustering models use exclusively information concerning the cluster closest to a data point. This results in a partition of the data space \mathbb{R}^p , into the so called Voronoi cells $\mathcal{P} = \{\mathcal{V}_1, \dots, \mathcal{V}_M\}$, with the cluster means $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k$ as determining points.

In a more formal way, let $S = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k\}$ be the set of the k -means cluster centres, then the Voronoi cell of the j -th center $\boldsymbol{\mu}_j$ is given by

$$\mathcal{V}_j = VR(\boldsymbol{\mu}_j, S) = \bigcap_{\boldsymbol{\mu}_l \in S \setminus \{\boldsymbol{\mu}_j\}} D(\boldsymbol{\mu}_j, \boldsymbol{\mu}_l), \quad (3.3.0.12)$$

$$D(\boldsymbol{\mu}_j, \boldsymbol{\mu}_l) = \{x \in \mathbb{R}^p : \|\boldsymbol{\mu}_j - x\|^2 < \|\boldsymbol{\mu}_l - x\|^2\}. \quad (3.3.0.13)$$

In other words, the Voronoi cell \mathcal{V}_j is the collection of all points in \mathbb{R}^p , that are closer to $\boldsymbol{\mu}_j$ than to any other center $\boldsymbol{\mu}_{j'}$, $j' \neq j$.

Using this cells we can rewrite the cluster distance and inverse cluster size score respectively, in terms of the Voronoi cells as

$$s(\mathbf{x}) = f \left(\sum_{j=1}^k \|\boldsymbol{\mu}_j - \mathbf{x}\|^2 \mathbf{1}_{\{\mathbf{x} \in \mathcal{V}_j\}} \right), \quad (3.3.0.14)$$

$$s(\mathbf{x}) = g \left(\sum_{j=1}^k \#\{i : \mathbf{x}_i \in \mathcal{V}_j\} \mathbf{1}_{\{\mathbf{x} \in \mathcal{V}_j\}} \right), \quad (3.3.0.15)$$

where $f(x) = 1 - e^{-x}$ and $g(x) = 1 - e^{-1/x}$.

Thus, in the case of cluster distance, we approximate the assignment function $\hat{H}(\mathbf{x})$ on the Voronoi cell by the euclidean distance to the cluster center. In the case of the inverse cluster size, the assignment function is simply given by a piecewise constant function, constant on each Voronoi cell.

The k -means methods KMD and KMC are related to density estimation through inverse cluster distance and cluster-size respectively, i.e. a monotone transformation of this properties could be used to estimate a density.

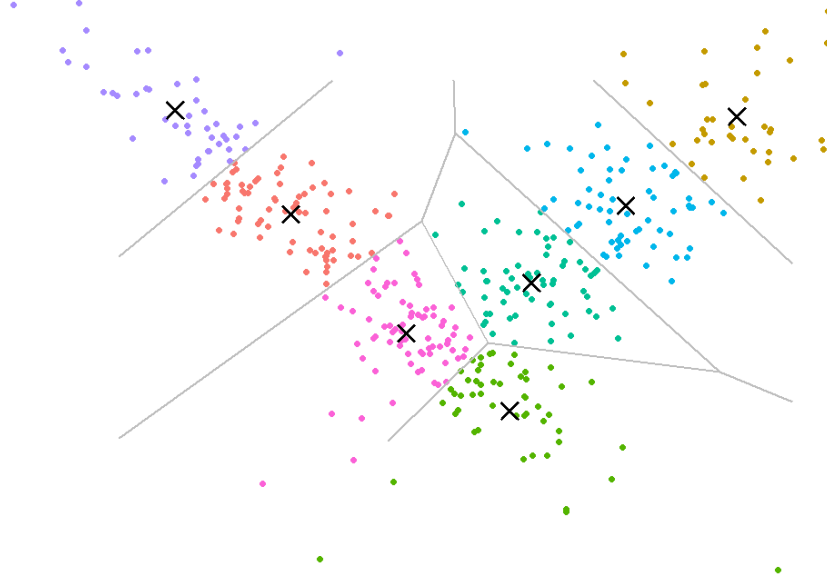


Figure 3.4: This figure shows the result of the k -means clustering algorithm on the butterfly dataset, using $k = 7$ clusters and the corresponding boundaries of the Voronoi cells.

Handling Different Datatypes

The k -means algorithm handles categorical/ordinal variables internally. This happens by extending the distance metric to categorical variables. Ordinal variables get converted to categorical variables, whereby the ordinal information gets lost. We use the notation $\mathbf{x} = (\mathbf{x}_{num}, \mathbf{x}_{cat})$, where $\mathbf{x}_{num} = (x_1, \dots, x_m)$ represents the numerical features and $\mathbf{x}_{cat} = (x_{m+1}, \dots, x_p)$ the categorical features, for some $j \in \{1, \dots, m\}$. Using this notation the new distance measure takes the form

$$d(x, x') = \sqrt{\sum_{i=1}^m (x_i - x'_i)^2 + \sum_{i=m+1}^p \mathbf{1}_{\{x_i = x'_i\}}} \quad . \quad (3.3.0.16)$$

But we face the problem that numerical values are potentially on a completely different scale i.e. distances in the numerical space can potentially have values of sizes in the 100s or 1000s and distances in the categorical space are limited by 1. To address this problem we normalize the data first such that both, numerical and categorical features, have approximately similar scales of distances. The normalization forces the numerical features to have mean 0 and variance 1:

$$\text{normalize}(\mathbf{x}^j) = \frac{\mathbf{x}^j - \bar{\mathbf{x}}^j}{\sigma(\mathbf{x}^j)}, \quad (3.3.0.17)$$

where $\sigma(\mathbf{x}^j)$ denotes the sample standard deviation of the feature vector \mathbf{x}^j .

Choice of Parameters

- **Number of k -means models in the ensemble** $n \in \mathbb{N} \setminus \{0\}$ (default $n = 5$):
The principle of ensembling k -means models is similar to the principle of bagging. Thus adding new models generally improves performance (for some time) but the fitting as well as the scoring time increases linearly in the number of models. Since there doesn't seem to be a big difference in performance for $n > 5$ models, we choose $n = 5$ as default.
- **Number of clusters in each model** $k \in \mathbb{N} \setminus \{0\}$ (default: $k = 200$):
The most important parameter for the k -means models is the number of clusters. For Novelty Detection scenario we choose it as a big number, because we want to capture the shape of the data as accurate as possible. However we shouldn't set k too big because then we risk, that the clustering adapts to noise.

Advantages

- The clustering algorithm can handle categorical data together with numerical data in a nice way with the mentioned adaptation of the distance measure.
- We can handle a big number of features by using feature subsampling and a big number of observations using observation subsampling.
- By using a big enough number of clusters k the method can also capture more complicate shapes of data even though it only consists of "sphere" -shaped clusters. Because we only care about distance to the center or number of points in the cluster respectively, it doesn't matter if a structure in the data that forms one complicated shaped cluster is represented by one or by several clusters. Thus the method is very adaptive to the data.
- The method takes advantage of bagging effects due to ensembling the diverse k -means clustering results. The resulting clusterings are inherently noisy due to the noisy fitting algorithm that can produce different results for different initializations and diverse because of feature and observation subsampling.

Disadvantages

- A major disadvantage of the algorithm are the comparatively many hyperparameters that we have to choose. In addition that problem gets intensified by the strong dependence of the methods performance on some of this parameters, e.g. the number of clusters k often has a big impact on the performance of the resulting detection performance.
- The clustering result heavily relies on the distance function we use and may be not very appropriate for complex high dimensional data. Nevertheless we can solve this problem in parts with using an appropriate colsample rate.
- If one uses too many clusters k the methods can run into severe problems in the case of a slightly contaminated dataset. In this case, the anomaly points could form their own cluster and thus decrease the performance in a significantly way or even dispose the method to perform worse than random.

3.4 Autoencoder

An Autoencoder (AE) is a neural network that is trained to reproduce as output its input. The idea of an Autoencoder is to use a neural network to condense the dimension of a dataset or otherwise constrict the capacity of it, while keeping as much information as possible.

We want the network to learn to predict the input data at the output layer, thus the target for prediction consists of the input data itself. Here we design the network to have a bottleneck-layer between the input and the output layer, which has fewer dimensions than the original data. By training such an Autoencoder we create at the "bottleneck"-layer a condensed representation of our data, from which can reconstruct the original data as good as possible. The idea of the Autoencoder is well captured by the book of [Goodfellow, Bengio, and Courville \(2016\)](#), chapter 14 and thus this section is based on the information of that book.

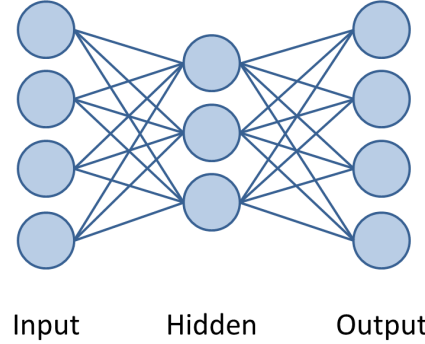


Figure 3.5: This figure shows a schematic picture of an Autoencoder, which reproduces the input at the output layer. In this example the input dimension, which is equal to the output dimension, has size four and there is only one hidden layer of size three. Note that the biases are missing for visual purposes.

This hidden layer ("bottleneck") \mathbf{h} generates the **code** as which the data is represented. The layer \mathbf{h} implies a natural decomposition of the Autoencoder map. We can decompose the mapping function from input to output in two parts, the **encoder** map $\mathbf{h} = f(\mathbf{x})$, that maps from the data to the code and the **decoder** map $\mathbf{r} = g(\mathbf{h})$, that attempts to reconstruct the original data from the code.

It is not the goal to learn the map $g(f(\mathbf{x})) = \mathbf{x}$ perfectly, because that is not useful. Rather we want to build a map, that reproduces the data as optimal as possible but with a restriction on the map that makes it impossible to do this perfectly. Thus, we restrict the capacity of the neural network either through a layer that has smaller dimension than the input or we impose regularization, which forces the Autoencoder to reproduce the data only approximately. The result is that the AE enforces a prioritization on the important aspects of the data and discards infrequent pattern and noise. These favoured aspects then turn out to be very useful for other tasks.

The idea of an anomaly score based on the Autoencoder is to use the reconstruction error of new observations, i.e. a measure of the difference between a new data point \mathbf{x} and its reconstruction $g(f(\mathbf{x}))$, as anomaly score.

Algorithm

To explain the functional principle of the Autoencoder we first have to define the notions of the neuron and the neural net.

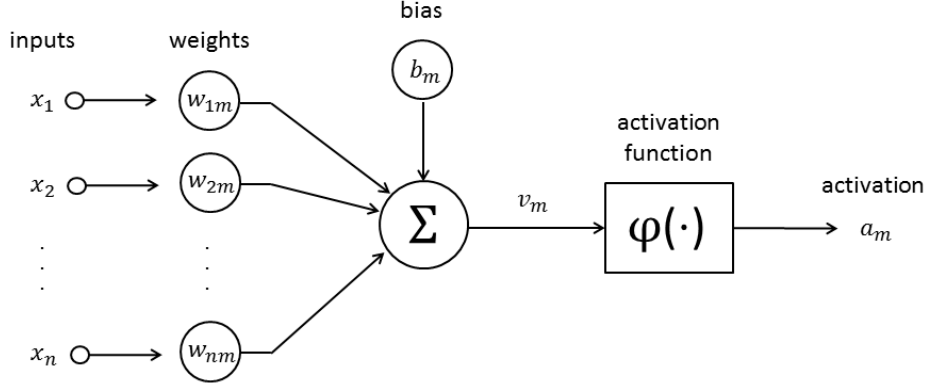


Figure 3.6: This figure shows the computational schema of a single neural network unit. The inputs, consisting of previous neuron activations and a bias, are combined by a weighted mean (weights w_{im}). The activation/output of the unit is produced by a transformation of the weighted mean with an activation function φ .

The basic building blocks of a neural network is the neuron, which is motivated by the biological neuron. The strengths of the synapses/connections are represented by weights w_{km} , where k denotes the index (in its layer) of the unit the signal originates from and m the index (in its layer) of the neuron, we want to calculate the activation. Further there is a so called bias added to activations, which is just an additive constant, in order to add more flexibility to the unit. To compute the activation of a unit the inputs are combined as a weighed sum and sent through an activation function $\varphi(\cdot)$ (see Figure 3.6):

$$a_m = \varphi \left(b_m + \sum_{i=1}^n w_{im} x_i \right) . \quad (3.4.0.1)$$

In a next step we combine this neurons in multiple layers to define a neural network which can exhibit different architectures (different number of layers, number of neurons per layer):

The Autoencoder neural networks are special cases of such neural networks. As mentioned above, the restriction of the capacity of an AE can take different forms:

- *Undercomplete Autoencoders*: This is the most popular from and simply restricts the dimension of the code \mathbf{h} to be smaller than the dimension of the input data \mathbf{x} .
- *Regularized Autoencoders*: In the case, one chooses a code dimension, which is equal or bigger than the input dimension. This is called the **overcomplete** case. The restriction of the capacity then happens through modifying the loss function, by penalization terms that favour certain properties of the code. This properties can be sparsity of the representation, smallness of the derivative of the representation, robustness to noise, robustness against missing inputs, etc.

- *Sparse Autoencoders*: We add a sparsity penalty, in form of an $L1$ norm of the parameters, to the loss function we optimize:

$$L(\mathbf{x}, g(f(\mathbf{x}))) + \lambda \|\mathbf{w}\|_1 \quad (3.4.0.2)$$

- *Denoising Autoencoders (DAE)*: We add some noise to the input data points such that the network learns to be robust against noise:

$$L(\mathbf{x}, g(f(\tilde{\mathbf{x}}))), \quad (3.4.0.3)$$

where $\tilde{\mathbf{x}} = \mathbf{x} + \epsilon$ is a noise corrupted version of the original data point.

- *Contractive Autoencoders (CAE)*: We add a derivative penalty term to the loss function to make the learned function robust against small changes of \mathbf{x} :

$$L(\mathbf{x}, g(f(\mathbf{x}))) + \lambda \|\nabla_{\mathbf{x}} \mathbf{w}\|_2^2. \quad (3.4.0.4)$$

The learning task for the (undercomplete) AE consists of minimizing the following loss function

$$L(\mathbf{x}, g(f(\mathbf{x}))) \quad , \quad (3.4.0.5)$$

which quantifies the severity of a deviation between \mathbf{x} and $g(f(\mathbf{x}))$. A common used definition for the loss function is the mean squared error $L(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2$.

We define the anomaly score of a new data point \mathbf{x} as a monotone function of the reconstruction error (3.4.0.5) we optimize the neural network on and thus verify how good the network has learned to reconstruct that type of data point.

$$s(\mathbf{x}) = L(\mathbf{x}, g(f(\mathbf{x}))) \quad . \quad (3.4.0.6)$$

The rationale behind this idea is that the Autoencoder learns to encode frequent, normal data and thus has a small reconstruction error for such data but produces a big one for infrequent, abnormal data.

Since the Autoencoder is simply a special case of a feed-forward neural network, it can be trained with the exactly same techniques. The concrete optimization of the layer connecting weights, which happens by minimizing the loss function (3.4.0.5), is not an easy task and modern optimization techniques use a number of tricks to achieve this. The main idea is to perform minibatch gradient descent on the gradients computed with the famous back-propagation algorithm. This algorithm allows to propagate an error beginning from the output layer back to the input layer by changing the weights accordingly.

As mentioned, this optimization needs a lot of tricks and heuristics to function properly, because the optimization objective is highly non-convex. Thus the gradient descent procedure can get stuck in a lot of (suboptimal) local minima which then can destroy the performance of the fit. We will only mention these methods but not go into detail, since the goal of this thesis is to investigate different anomaly detection methods. The tricks include:

- *Stochastic Gradient Descent (SGD)*: Each gradient descent update is based only on a small batch/subset of all training data, in order to save time, because a complete update is computationally quite expensive.

- *Adaptive gradient method (ADAGRAD)* (Duchi, Hazan, and Singer, 2011): The idea of adaptive gradient method is to adapt the learning rate for each parameter based on the previous updates. We adapt the learning rate such that larger updates are performed on infrequent parameters and smaller updates on frequent. This is done by setting the learning rate at time $t + 1$ equal to an initial learning rate divided by the sum of squares of the previous gradients up to time t .
This procedure is especially suited for sparse data and is proven to increase robustness, especially in the case of large scale neural networks.
- *Momentum* (Qian, 1999): Momentum is a concept that was invented to solve the problem of navigating through ravines, i.e. regions in which the surface changes a lot faster in one dimension than in another dimension. This occurs frequently in the case of local optima. In such situations the updates oscillate in the directions of strong curvature, while making small progress in the direction of weaker curvature. To fix this the method accelerates SGD by adding in each update step a fraction of the last time's update step to the current update, which dampens oscillations and accelerates convergence.
The method can well be visualized by letting a ball rolling down a hill, where it accumulates momentum (thus the term) as it rolls downhill and increases velocity. The same happens for the gradient updates, if they change directions momentum can't build up but if the update directions face approximately in the same direction, it can accumulate.
- *Nesterov Accelerated Gradient (NAG)* (Nesterov, 1983): Nesterov gradient acceleration attempts to solve problems which we run into, by blindly following the slope the ball follows when we push it down a hill. For example if the ball accumulates momentum rolling down a hill and its momentum is too big near the minima, it can happen that it rolls out of the minima and never returns to it again. NAG solves that problem by adding some smartness to the ball that allows him to "look forward" in the direction it is moving. This allows the ball to slow down before a hill slopes up again.
The method uses the momentum direction to look forward "where" the ball is moving. This method then measures the gradient at this "extrapolated" point and makes a correction using this information.
- *Weight Dropout* (Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov, 2014): Since deep neural networks often contain a huge number of parameters, they are very powerful in representing and adapting to data, which makes overfitting a considerable problem.
It can be shown that dropout is a computational efficient way to implicitly average different neural networks without having to train a number of different networks at a time. Dropout works by randomly dropping units with a certain probability (typically $p = 0.5$) after each training iteration. Dropping a unit means temporarily removing it from the network along with all its incoming and outgoing connections. This avoids co-adaptation of units, reduces the speed of overfitting and thus often drastically improves performance.
- *L1 or L2 regularization*: The idea of $L1$ or $L2$ regularization is to add an $L1$ penalty $c \cdot \|\theta\|_1$ or an $L2$ penalty $c \cdot \|\theta\|_2$ respectively, to the training criterion of the neural network. This results in sparse weights in the case of $L1$ regularization and in overall shrunken weights in case of $L2$ regularization.

- *early stopping*: The method of early stopping is a general method for iteratively learning supervised learning methods to determine the optimal number of iterations. Supervised learning methods that adapt closer to the training data in each iteration, generally improving performance on new data for a certain time until they get worse again because of overfitting. Early stopping tries to find this point by using a holdout set to monitor the performance and stops the fitting process if the performance doesn't increase again for a certain number of iterations.

The complexity of the data representation of the Autoencoder can be extended in different ways. An interesting way to increase complexity, is to add more layers between the input and the bottleneck layer and the bottleneck and output layer respectively (see Figure 3.7).

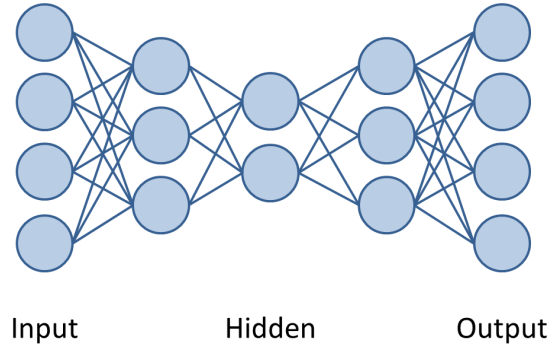


Figure 3.7: This figure shows a schematic picture of a deeper Autoencoder than Figure 3.5, with input/output dimension 4 and three hidden layers with sizes of 3, 2 and 3.

Another way to increase the complexity of the AE is to build a so called Deep Autoencoder. The idea is to train an AE, propagate the samples through to the bottleneck and use these activations as the input for another AE, as described in [Bengio et al. \(2013\)](#).

Theoretical Modelling Approach

As described in the algorithm section, the scoring function is defined as a monotone and continuous transformation of the AE reconstruction error,

$$\hat{H}(\mathbf{x}) = s(\mathbf{x}) = L(\mathbf{x}, g(f(\mathbf{x}))), \quad (3.4.0.7)$$

with encoding and decoding maps f and g . Because of the complicated structure of these maps, it is difficult to say how the estimation function effectively looks like. One thing one can show about the scoring function, is its continuity in \mathbf{x} if one chooses the activation function φ to be continuous. In our experiments this holds true, because we choose the continuous *tanh* activation function.

Another thing, that we know about the Autoencoder error, is that the reconstruction $g(f(\mathbf{x}))$ lies on a lower dimensional manifold in \mathbb{R}^p , with the dimension of the bottleneck layer. Therefore, the anomaly score of a point \mathbf{x} is defined as the (euclidean) distance of the point to its embedding of the AE in the original space $g(f(\mathbf{x}))$.

Thus, the scoring function is related to density estimation through this manifold, i.e. the euclidean distance of a point to its embedding in the manifold, could be used to define a density function. For an example of such an embedding consult Figure 4.2.

Choice of Parameters

- **Hidden layer architecture:**

Experiments with different architectures showed, that a one-layer architecture with $q \in \mathbb{N} \setminus \{0\}$ neurons gives the best performance. The default hidden layer size we choose as a factor of the dummy dimension, which makes sense since the categorical features are dummy encoded at the input layer of the AE. Experiments showed that a (compression) factor of 0.5 gives often good results (i.e. choose $q = 0.5 \cdot$ dummy dimension).

- **Number of training epochs** $e \in \mathbb{R}_{>0}$ (default: $e = 5.0$):

Choosing the number of epochs equal to 5.0, i.e. training the AE on a number of samples equal to the number of all training samples seems to work well.

- **Activation function** (default: *tanh*):

This parameter describes which activation function $\varphi(\cdot)$ we use in the neurons. Testing different activation functions showed that *tanh* delivered the best results.

Handling Different Datatypes

The AE can nicely deal with categorical features by interpreting them in a dummy encoded way. This adds a few more weights to the neural network and thus doesn't increase the training process time by a lot.

Advantages

- The method is purely data driven, i.e. does not make any assumptions about the generative process of the data.
- The method allows to capture complex relations in the normal data and to distinguish them from the relations in the anomalous data.
- The handling of categorical variables as dummy variables gives good results for this method.

Disadvantages

- If one categorical feature consists of a huge number of levels, one can potentially run into problems. This is the case, because then the dummy dimension is huge and thus, the input dimension of the AE is too big, meaning that there are too many weights to optimize. To solve this one should restrict the number of levels per categorical feature to a certain number.

3.5 Feature Regression and Classification

The anomaly detection method Feature Regression and Classification (FRaC) has been proposed by the paper of [Noto et al. \(2012\)](#) and uses an approach, that isn't well known in anomaly detection so far, the feature-modelling anomaly detection approach. The idea of the algorithm is to build prediction models for each feature, while using the rest as predictors. Then the idea behind the anomaly detection technique is, that the feature models capture the (potentially complex) relations between the different features which hold for the normal data but do not hold for the anomalous data. These differences in relations for the anomalies then get exploited to define an anomaly score.

Anomaly Assumption

We have to assume, that the data of the normal class has a distribution that can be learned well enough by our classification and regression models. Further we assume, that the normal distribution differs enough from the distribution of the anomaly class. In this case the prediction models produce big enough errors if they try to predict anomaly feature values with models trained on normal data.

Algorithm

The basic idea is to build an internal prediction model of the normal data, which has another distribution than the anomalous data. Thus the model produces big errors when comparing the prediction to the true values of the features for anomalous samples.

In more detail this means we build for every feature a prediction model that predicts that feature, using all the other features in the set. Then we use the difference of the predicted values and the true value as a feature-wise anomaly measure. Combining the errors of the prediction models for each feature, we get one anomaly score for each sample.

For the concrete implementation of the FRaC-algorithm we need to specify three major points:

- *Choice of features:* Without any knowledge about the data there is no basis to decide which features should be chosen for our anomaly algorithm. Thus we build a prediction model for each feature. But we introduce one exception, we don't build models for almost constant features. The reason for this is that they tend to give data points for which that feature doesn't have the same value as almost all other data points, a huge anomaly score and thus can destroy the whole performance of the algorithm. Therefore we don't build models for features with a variance smaller than 10^{-4} .

Since we choose prediction algorithms that decide themselves which predictors they want to include in the prediction model, we just choose all the remaining features as predictors.

- *Choice of prediction algorithm:* We can choose any algorithm that matches the type of the feature we want to predict, i.e. for numeric variables we can choose any regression model and for categorical variables we can choose any classification model.

- *How to combine feature models:* We combine the features using the normalized surprisal score.

Let $\mathbf{x} = (x_1, \dots, x_p)$ be a p -dimensional data point. Then we define the projection on the $(p-1)$ -dimensional subspace that throws away the j -th dimension, $i \in \{1, \dots, p\}$ by

$$\rho_j(\mathbf{x}) = (x_1, \dots, x_{j-1}, x_{j+1}, \dots, x_p) \quad . \quad (3.5.0.1)$$

Further let $\mathbf{t} = (t_1, \dots, t_p)$ be the spaces the features, e.g. $t_i = \mathbb{R}$ for numerical or $t_i = \{\text{category } 1, \dots, \text{category } c\}$ for categorical features. Then we denote the j -th prediction model as $C_j : \rho_j(\mathbf{x}) \rightarrow t_j$. Using this definitions, an anomaly score based on the feature prediction model errors is a map $f(\mathcal{C}, \mathbf{x}) \rightarrow \mathbb{R}$.

We denote by $P(x_j | C_j, \rho_j(\mathbf{x}))$ the likelihood of an observed value x_j given the feature model C_j and the remaining features $\rho_j(\mathbf{x})$. This conditional likelihood still needs to be estimated properly.

For the *feature regression and classification* approach the authors of (Noto et al., 2012) propose to use an information theoretic approach to measure the amount of evidence, that a data point is an anomaly. We measure this evidence by the amount of surprisal, which is given by

$$\text{surprisal}(p) = -\log(p) \quad . \quad (3.5.0.2)$$

Then we define the anomaly score of an instance by

$$s(\mathbf{x}) = \sum_{j=1}^p \text{surprisal}(P(x_j | C_j, \rho_j(\mathbf{x}))) \quad . \quad (3.5.0.3)$$

Since the probability of a feature value x_j of a new point \mathbf{x} , $P(x_j | C_j, \rho_j(\mathbf{x}))$ can't be defined directly using C_j , we involve cross validation.

We use the predictions of the validation folds to compute a set A_j of (observed, predicted) values for the feature j . Then an error model E_j is built to model the probabilities of prediction errors (of the value pairs) in A_j .

Thus the FRaC model defines the probability $P(x_j | C_j, \rho_j(\mathbf{x}))$ in terms of E_j , $C_j(\rho_j(\mathbf{x}))$, but the way to define this probability depends on the specific feature type:

- *numerical:* The error between the real and the predicted feature value is defined as $x_j - C_j(\rho_j(\mathbf{x}))$. This errors are computed for all training samples using CV, then the range of the errors is binned into \sqrt{n} bins, the number of error values per bin is counted and this counts get normalized (dividing by n). The result is a probability distribution on the range of error values. Since the choice of the bins is in principle arbitrary and there are potentially boundary effects that can distort the distribution we smooth the (step-) probability distribution using a Gaussian kernel with standard deviation of one bin length.

The probability $P(x_j | E_j, C_j(\rho_j(\mathbf{x})))$ is therefore defined as the mass of the bin, that corresponds to an error of size $x_j - C_j(\rho_j(\mathbf{x}))$.

- *categorical:* For categorical features, the error distribution E_i is defined as the confusion matrix entry. This means if the FRaC predicts the j -th feature value y to be g , then the error model returns the frequency of such (y, g) pairs in A_j .

This last described error model for categorical variables is applicable to any classification method. But since we use the random forest in our experiments in section 5, we use instead of this just described error model, the internal error model of the random forest itself. The random forest computes as a by-product a probability estimate for every classification prediction.

With this way to calculate $P(x_j|E_j, C_j(\rho_j(\mathbf{x})))$, we can now define the *normalized surprisal score*, which is given by

$$NS(\mathcal{C}, \mathbf{x}) = \sum_{m=1}^M \sum_{j=1}^p \text{surprisal}(P(x_j|C_j, \rho_j(\mathbf{x})) - \text{entropy}(\{x_{1j}, \dots, x_{nj}\})), \quad (3.5.0.4)$$

where the entropy of a set \mathcal{S} is given by

$$\text{entropy}(\mathcal{S}) = \sum_{v=1}^k -p_v \log(p_v), \quad (3.5.0.5)$$

v representing to the k distinct values in set \mathcal{S} and p_v the proportion of elements in \mathcal{S} that have the same value.

Theoretical Modelling Approach

In this section we stick to the special case of a dataset with only numerical features. Let us first discuss the score of a regression feature model of the numerical feature j . We denote the prediction of the feature model C_j at the point \mathbf{x} as $\hat{x}_j = C_j(\rho_j(\mathbf{x}))$. As described before, we denote the error model of the j th feature as E_j . Further, the error model for regression models are normalized histograms with $b = \lfloor \sqrt{n} \rfloor$ bins A_{1j}, \dots, A_{bj} . Thus we can write the estimated probability that the j th feature value of \mathbf{x} is equal to x_j , as

$$p_j := p_j(\mathbf{x}) := P(x_j|E_j, C_j(\rho_j(\mathbf{x}))) = \sum_{k=1}^b \alpha_{jk} \mathbf{1}_{A_{kj}}(x_j - \hat{x}_j), \quad (3.5.0.6)$$

where α_{jk} is the estimated probability of an error being in the j th bin. We recognize, that the estimated probability function is a step function. Thus we can rewrite the normalized surprisal score for all features, using a single prediction model type m , as

$$\begin{aligned} s(\mathbf{x}) &= NS(\mathbf{x}) \\ &= \sum_{j=1}^p (\text{surprisal}(p_j) - \text{entropy}(\{x_{1j}, \dots, x_{nj}\})) \\ &= \sum_{j=1}^p p_j \log(p_j) - \sum_{j=1}^p \text{entropy}(\{x_{1j}, \dots, x_{nj}\}) \\ &= \log \left(\prod_{j=1}^p p_j^{p_j} \right) - \sum_{j=1}^p \text{entropy}(\{x_{1j}, \dots, x_{nj}\}). \end{aligned} \quad (3.5.0.7)$$

We recognize that combining the formulas (3.5.0.6) and (3.5.0.7), the scoring function stays constant as long as we keep staying in the same combinations of bins $A_{k_1,j}, \dots, A_{k_j,j}$. This implies that the space can be divided in $N = b^p$ regions $\mathcal{R}_1, \dots, \mathcal{R}_N$ where the anomaly score is constant. The shape of the regions depends on the relation of $C_j(\rho_j(\mathbf{x}))$ with

Algorithm 14 $FRaC(X, n_{folds})$

Inputs: $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ - input data**Output:** anomaly scoring function $s(\mathbf{x})$

```

1: Initialize  $FRaC\_model$ 
2: for each feature  $j$  in  $X$  do
3:   for feature prediction model  $m \in \{1, \dots, M\}$  do
4:      $A_{m,j} \leftarrow \emptyset$ 
5:     // will be list of (observed, predicted) value pairs, used to error model to
6:     // estimate  $P(x_j|C_j, \rho_j(\mathbf{x}))$ 
7:     for each CV-fold  $f \in \{1, \dots, n_{folds}\}$  do
8:        $T_f, V_f \leftarrow \text{divide}(X, f)$ 
9:       // split into training set  $T$  and Validation set  $V$  with
10:      // proportions  $((n_{folds} - 1)/n_{folds}, 1/n_{folds})$ 
11:       $valset_f \leftarrow (\rho_j(\mathbf{x}_i), x_{ij})$  for each  $\mathbf{x}_j \in V_f$ 
12:       $trainset_f \leftarrow (\rho_j(\mathbf{x}_i), x_{ij})$  for each  $\mathbf{x}_j \in T_f$ 
13:       $C_{p,i,f} \leftarrow \text{train}_p(trainset_f)$ 
14:      // learn prediction model for feature  $j$  with model  $m$ 
15:       $A_{m,j} \leftarrow A_{m,j} \cup (x_{ij}, C_{m,j,f}(\rho_j(\mathbf{x}_i)))$  for each  $\mathbf{x}_i \in valset_f$ 
16:    end for
17:     $E_{m,j} \leftarrow \text{error\_model}(A_{m,j})$ 
18:    // model error distribution of  $A_{m,j}$  (depends on model and feature type)
19:     $trainset \leftarrow (\rho_j(\mathbf{x}_i), x_{ij})$  for each  $\mathbf{x}_j \in X$ 
20:     $C_{m,j} \leftarrow \text{train}(trainset)$ 
21:    // final predictor trained on entire  $X$  for predictions on new datapoints  $\mathbf{x}$ 
22:  end for
23: end for

```

$$\begin{aligned}
s(\mathbf{x}) &:= NS(\mathcal{C}, \mathbf{x}) \\
&= \sum_{m=1}^M \sum_{j=1}^p \text{surprisal}(P(x_j|E_m, C_{m,j}(\rho_j(\mathbf{x}))) - \text{entropy}(\{x_1, \dots, x_n\})
\end{aligned}$$

18: **return** $s(\mathbf{x})$

its argument $\rho_j(\mathbf{x})$. The region boundaries adapt to the shape of the prediction function C_j . As an example, if we use random forest, the regions take the shape of axis parallel rectangles and thus the scoring function takes the form of a step function.

This method can as well be related to density estimation, through the distances of the predicted feature values and the real ones, i.e. we can use the inverse distances as a basis for a density estimation. Note that the "distance" is not defined in an euclidean sense, but by the error model used for the different feature types, i.e. inverse error histogram proportion in case of regression or inverse confusion matrix frequency in case of classification.

Handling Different Datatypes

The handling of different data types is explicitly discussed in the algorithm section before. Different data types are basically treated differently by using the matching supervised learning algorithm (regressors for numerical features, classifiers for categorical features) and the right error model (histogram error distribution for numerical features, confusion table for categorical features). Further the method can be adapted to any data type that a supervised learning algorithm can learn and for which an appropriate error model can be defined.

Choice of Parameters

- *Choice of base learner and its parameters:*

As mentioned before, arbitrary regression and classification methods can be used for this method. The authors of the FRaC paper (Noto et al., 2012) proposed to use an ensemble of SVM's with different kernel functions and decision trees. In our experiments we only used the random forest, but for further investigations it will be completely reasonable to use an ensemble of methods including for example boosted trees, neural networks, linear models (such as Lasso) or SVM's. Further, experiments show that the method performs better when we use simpler models, e.g. choose a small number of trees for the random forest.

Advantages

- The method handles numerical and categorical data in an explicit and balanced way using the notion of surprisal.

Disadvantages

- One needs to control features that are almost constant, because their prediction model tends to be a simple constant. This has the effect, that the error model assigns the few values that deviate from this prediction a very high surprisal score. This can destroy the performance of the method.
- One often achieves only a moderate performance on original data, especially if the anomaly signal is only present in a few features. Thus it often works better with en-

coded data, because in this case all features tend to deliver a relevant to contribution to the score.

- The algorithm can be slow depending on the number of features, the dataset size and especially the type or implementation of the supervised learning algorithm. In our experiments we used the random forest classifier/regressor as supervised learning algorithm which makes the implementation of the method quite slow, but of course there are faster methods/implementations available.

3.6 One-Class SVM

The basic idea of the one-class support vector machine (OSVM) (Tax, 2001) is probably the most straight forward one of all described algorithms in this thesis. Conceptually, the method draws a line around the data points, while leaving as few space as possible between the boundary and the data points. In addition, the flexibility of the boundary gets restricted to keep the boundary simple and further allow the algorithm to violate the boundary condition for a small number of points, i.e. some are allowed to lie outside of the boundary.

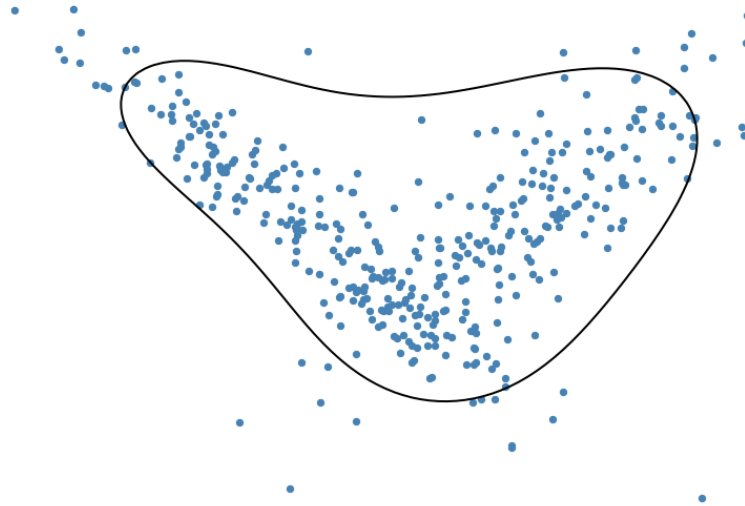


Figure 3.8: This scatter plot shows the already seen butterfly dataset (see Figure 3.1). The black line indicates the decision boundary computed by the OSVM model. Points that lie inside this boundary are considered normal whereas points that lie outside, are considered anomalies.

The model is formulated in a similar form as the Support Vector Machine and due to that similarity, the model is called one-class SVM.

Anomaly Assumption

The anomaly assumptions for the one-class SVM are:

- *Low density assumption:* We assume that outliers occupy low density regions of the data space.
- *Kernel assumption:* We assume that the high density regions of the data can be characterized by the kernel model.

Algorithm

Similar to the well known two-class classification SVM we begin first with a very primitive type of model and add complexity and adaptability with slack variables and kernel functions afterwards.

As a first step, we describe the data with a simple hypersphere. Thus the optimization problem consists of solving the following *structural loss* function with constraints:

$$L_{struct}(R, a) = R^2, \quad \text{s.t.} \quad \|\mathbf{x}_i - \mathbf{a}\|^2 \leq R, \quad \forall i. \quad (3.6.0.1)$$

To allow for some robustness we relax the constraints that the distance of the data points \mathbf{x}_i to the center \mathbf{a} has to be strictly smaller than R and add penalties for this violations. This way, we introduce an *empirical loss* part to the loss function, in form of the slack variables $\xi_i, \xi_i \geq 0, \forall i$. The combined loss function is written as

$$L(R, \mathbf{x}, \xi) = \underbrace{R^2}_{L_{struct}} + C \underbrace{\sum_i \xi_i}_{L_{emp}}, \quad \text{s.t.} \quad \|\mathbf{x}_i - \mathbf{a}\|^2 \leq R + \xi_i, \quad \xi_i \geq 0, \forall i, \quad (3.6.0.2)$$

where C is the trade-off parameter that balances the (volume based) structural error and the (violation distance based) empirical error.

The problem formulation of (3.6.0.2), which is known as the "Primal Problem" definition of the SVM gets rephrased into the equivalent "Dual Problem" which makes it straight forward to solve it with quadratic programming.

A data point \mathbf{x} gets classified as an anomaly, if it has a distance bigger R to the center \mathbf{a} , i.e. if $\|\mathbf{x}_i - \mathbf{a}\|^2 \leq R$. To achieve a gradual score, one can define an anomaly score using $\|\mathbf{x}_i - \mathbf{a}\|^2$ and R .

In a next step we approach the problem, that the hypersphere is a lot too restrictive model to capture the shape of real data. Thus, we want to add flexibility to the data description model of the one-class SVM. We achieve this in the standard way used to add flexibility to an SVM model, by using a kernel function ("kernel-trick").

We map the data into a different space with the mapping Φ , i.e. $\mathbf{x}^* = \Phi(\mathbf{x})$. Therefore, the complete discussion of the problem formulation can be repeated with replacing all data points \mathbf{x}_i by $\Phi(\mathbf{x}_i)$. Since the optimization objective in the before mentioned "Dual Problem" formulation, only involves the data points in terms of inner products $\mathbf{x}_i \cdot \mathbf{x}_j$ which are then replaced by $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$, we can simplify this further with a kernel function

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \quad . \quad (3.6.0.3)$$

At the end, this results in the following "Dual Problem" formulation:

$$L(\alpha) = \sum_i \alpha_i K(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i,j} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \quad \text{s.t.} \quad 0 \leq \alpha_i \leq C \quad \forall i \quad , \quad (3.6.0.4)$$

where L should be minimized with respect to α . Then the decision function for the anomaly classification of a new data point \mathbf{x} is

$$f(\mathbf{x}) = \mathbf{1} \left\{ \underbrace{K(\mathbf{x}, \mathbf{x}) - 2 \sum_i \alpha_i K(\mathbf{x}, \mathbf{x}_i) - \sum_{i,j} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)}_{=: v(\mathbf{x})} \leq R^2 \right\} \quad (3.6.0.5)$$

As before, we can define a gradual anomaly score $s(\mathbf{x})$ using the function $v(\mathbf{x})$ and radius R .

Fortunately, the introduction of a kernel barely introduces computational costs, because the quadratic optimization problem remains unchanged in its complexity. The only extra cost is added, by the computation of the kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$.

Different kernels have been introduced for SVM's so far like the Gaussian/radial basis function (RBF) kernel, the polynomial kernel or the sigmoid kernel. The above mentioned work by Tax (2001) tested the two first kernels and concluded, that the Gaussian kernel is the favoured one for the one-class SVM. The advantages of the Gaussian kernel are its flexibility of the described boundary, the option to control the complexity of the boundary with the parameter σ and its tendency to produce a tighter description of the data, compared with other kernels.

The Gaussian kernel is defined as

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right) \quad , \quad (3.6.0.6)$$

where σ is the width parameter, which determines the flexibility of the SVM boundary. Different choices of σ produce diverse data models:

- For *very small* σ , the one-class SVM is identical to a Parzen density estimation.
- For *moderate* σ , the one-class SVM is identical to a weighted Parzen density estimation.
- For *very big* σ , the one-class SVM boundary becomes a rigid hypersphere.

Theoretical Modelling Approach

As described before we use the function $v(\mathbf{x})$, defined in equation (3.6.0.5), to define the anomaly score of a point \mathbf{x} . Further this distance describes the (squared) distance to the middle point of the hypersphere surrounding the data points in the transformed space. Formally this distance can be written, using the "Primal Problem"-notation, as

$$v(\mathbf{x}) = \|\Phi(\mathbf{x}) - \Phi(\mathbf{a})\|^2. \quad (3.6.0.7)$$

Thus, the anomaly detection scoring function is a transformation of the squared distance from a point to the center of the hypersphere in the transformed space $\Phi(\mathbb{R}^p)$.

The OSVM is related to density estimation through that distance defined in (3.6.0.7). Thus we could use a monotone transformation the inverse of this distance, to define a density.

Handling Different Datatypes

The handling of categorical and ordinal features happens through dummy encoding. Ordinal features get interpreted as categorical features with the ordinal information getting lost. But since the one-class SVM is specifically designed for continuous data, it makes often more sense to perform a representation step and then applying the algorithm to the encoded features.

Choice of Parameters

- **Kernel coefficient** $\gamma \in \mathbb{R}_{>0}$
(default: $\gamma = 1/p$):
Unfortunately the implementation of the OSVM doesn't use the parameter σ directly, rather it uses γ as a parameter, which is just one half of its reciprocal value: $\gamma = \frac{1}{2\sigma}$. As described in the algorithm section, this parameter determines how flexible the OSVM boundary is allowed to be.
- **Upper bound training errors / lower bound SV-fraction** $\nu \in (0, 1]$
(default: $\nu = 0.1$):
In the case of the Gaussian kernel function, the two quantities C and ν have a reciprocal relation: $C = \frac{1}{\nu n}$. The higher we choose C the less deviation from the model is allowed and vice versa. Thus choosing a lower ν value results in a stricter description, meaning less training values are allowed to lie outside the boundary.
- **Maximal number of training samples** $max_samples \in \mathbb{N}_{>0}$
(default: $max_samples = 10\,000$):
Because of the slow fitting algorithm of the OSVM we need to restrict the number of samples used to fit the algorithm, to compare the method with other methods. Thus we used this upper bound on the number of samples to restrict the fitting time of the algorithm. If more training samples are provided than this number, the algorithm randomly chooses a number of $max_samples$ samples to train on.

Advantages

- The method has the conceptually simple idea of classifying a single class.
- There is an explicit form of regularization steered through the ν parameter which specifies the bound on the number of support vectors. The bigger the number of support vectors, the more complicated the boundary can get.
- The method is very robust against noise features (see section 5.8).

Disadvantages

- The method is designed for continuous numerical data. Thus interpreting categorical data in a dummy encoded way is not always appropriate.
- The computational complexity and thus the runtime of the algorithm is very expensive and thus not useful for a lot of training data points.
- One needs to choose two main parameters ν and σ , which effect the class boundary in a significant way. Unfortunately it is a priori not clear, how to choose them for a general dataset.

3.7 Least Squares Anomaly Detection

The idea of the LSAD method (Quinn and Sugiyama, 2014) is very similar to the idea of the one-class SVM. The method models the probability density of the training data. Points that lie in a region with small density, are defined as anomalies.

The LSAD assumes a parametric model for the probability density and optimizes the parameter with respect to the squared loss of the density estimation (thus the term "Least Squares"). To capture complicated real life datasets, the method provides flexibility by using a kernel function, similar to the one-class SVM.

Anomaly Assumption

The anomaly assumptions are the same as the ones for the one-class SVM:

- *Low density assumption:* We assume that outliers occupy low density regions of the data space.
- *Kernel assumption:* We assume that the high density regions of the data can be characterized by the kernel model.

Algorithm

We denote our (labeled) training data as $\{\mathbf{x}_i, y_i\}_{i=1}^N$, where \mathbf{x}_i indicates the data points of our data set and y_i a provided label for different classes in the training data. We define the set of classes as $\mathcal{Y} = \{1, \dots, c\}$. Note that this label is not necessary, i.e. if there aren't different groups given in the data we just choose it to be the same group for all training data $y_i = 1, i = 1, \dots, n$.

The modelling step consists of estimating the class-conditional probabilities $p(y = i|\mathbf{x})$ by functions

$$q(y = i|\mathbf{x}, \theta_i) = \theta_i^T \phi(\mathbf{x}), \quad \text{for each } i \in \mathcal{Y} \quad (3.7.0.1)$$

where $\theta_i = (\theta_{i,1}, \dots, \theta_{i,B})$ for a number of parameters B . We can use $B = N$ if we want to use kernel basis functions at every point or $B < N$ for a random subset of training points. Let $\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_B}$ be the selected points, then the general transformation function is given by

$$\phi(\mathbf{x}) = (K(\mathbf{x}, \mathbf{x}_{i_1}), \dots, K(\mathbf{x}, \mathbf{x}_{i_B})) \in \mathbb{R}^B, \quad (3.7.0.2)$$

where we choose the famous squared exponential kernel for the LSAD:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{1}{\sigma^2} \|\mathbf{x} - \mathbf{x}'\|^2\right). \quad (3.7.0.3)$$

We fit the model using the squared loss for each class $i \in \{1, \dots, c\}$

$$J_i(\theta_i) = \frac{1}{2} \int (q(y = i|\mathbf{x}, \theta_i) - p(y = i|\mathbf{x}))^2 \quad (3.7.0.4)$$

Rewriting with this loss functions using Bayes law, we get

$$J_i(\theta_i) = \frac{1}{2} \int (q(y = i|\mathbf{x}, \theta_i))^2 - \int q(y = i|\mathbf{x}, \theta_i) p(\mathbf{x}|y = i) p(y = i) d\mathbf{x} + C. \quad (3.7.0.5)$$

Using an empirical approximation for this losses, ignoring the constant C and including an ℓ^2 -regularizer, we arrive at the following training criterion:

$$\hat{J}_i(\theta_i) = \frac{1}{2} \theta_i^T \Phi^T \Phi \theta_i - \theta_i^T \Phi \mathbf{m}_i + \frac{\rho}{2} \|\theta_i\|^2 \quad , \quad (3.7.0.6)$$

where $\Phi := (\phi(\mathbf{x}_1), \dots, \phi(\mathbf{x}_n))^T$ ($n \times B$ matrix) and $\mathbf{m}_i = (\delta_{y_1,i}, \dots, \delta_{y_n,i})^T$ is defined as a column vector that indicates the membership of class i , i.e. the j th entry is equal to 1 if the j th sample belongs to the class i ($y_j = i$) or 0 otherwise.

This loss function gets minimized by the following expression,

$$\hat{\theta}_i = (\Phi^T \Phi + \sigma \mathbf{I}_B)^{-1} \Phi \mathbf{m}_i \quad . \quad (3.7.0.7)$$

The nature of the estimator can cause estimates of the priors to be negative. We fix this by rounding up the estimates to zero in such cases,

$$q(y = i | \mathbf{x}, \hat{\theta}_i) = \max \left(0, \hat{\theta}_i^T \phi(\mathbf{x}) \right) \quad . \quad (3.7.0.8)$$

A prior estimate is then computed by normalizing over all classes,

$$\hat{p}(y = i | \mathbf{x}) = \frac{q(y = i | \mathbf{x}, \hat{\theta}_i)}{\sum_{j \in \mathcal{Y}} q(y = j | \mathbf{x}, \hat{\theta}_j)} \quad . \quad (3.7.0.9)$$

The computation of these (consistent) estimators is very fast, since we find the global optimum in a single step, where no iterative parameter search is required.

The anomaly model idea then happens by considering other classes $\{c+1, c+2, \dots\}$, which appear in the test but not in the training set. We denote these classes by $y = *$, $* \notin \mathcal{Y}$. As mentioned above we make the assumption, that outliers lie in low-density regions of the data space and that the high-density regions can be appropriately represented by the kernel model.

To obtain an anomaly detection method we finally need to define an estimator $\hat{p}(y = * | \mathbf{x})$ which is estimated by data which only contains data from classes in \mathcal{Y} . We estimate the conditional probability of an outlier $p(y = * | \mathbf{x})$ by

$$q(y = * | \mathbf{x}, \theta_*) = 1 - \theta_*^T \phi(\mathbf{x}) \quad . \quad (3.7.0.10)$$

Thus the process of calculating the anomaly detection model consists of learning the parameter θ_* . We want the last equation (3.7.0.10) to be small in high-density regions and big in low-density regions respectively, which is captured by the first part of the following loss function:

$$J_*(\theta_*) = \frac{1}{2} \int \left(1 - \theta_*^T \phi(\mathbf{x}) \right)^2 p(\mathbf{x}) d\mathbf{x} + \frac{\rho}{2} \|\theta_*\|^2 \quad . \quad (3.7.0.11)$$

The second part of the equation, which is basically a regularizer, forces the equation (3.7.0.10) to approach zero in low-density regions of the training data more quickly. This loss function (3.7.0.11) is similar to the loss function of the one-class SVM.

The empirical approximation of this loss function is

$$\widehat{J}_*(\theta_*) = \frac{1}{2} + \frac{1}{2} \theta_*^T \Phi^T \Phi \theta_* - \sum_{i=1}^n \theta_*^T \phi(\mathbf{x}_i) + \frac{\rho}{2} \|\theta_*\|^2 \quad , \quad (3.7.0.12)$$

and is optimized by

$$\widehat{\boldsymbol{\theta}}_* = \sum_{j \in \mathcal{Y}} \widehat{\boldsymbol{\theta}}_j \quad . \quad (3.7.0.13)$$

Finally, the estimated conditional probability of an outlier is given by

$$q(y = * | \mathbf{x}, \widehat{\boldsymbol{\theta}}_1, \dots, \widehat{\boldsymbol{\theta}}_c) = \max \left(0, 1 - \sum_{j \in \mathcal{Y}} q(y = j | \mathbf{x}, \widehat{\boldsymbol{\theta}}_j) \right) . \quad (3.7.0.14)$$

Thus, the anomaly score is defined as the estimated probability of being in the anomaly class, given the data point. The score takes the following form:

$$s(\mathbf{x}) = q(y = * | \mathbf{x}, \widehat{\boldsymbol{\theta}}_1, \dots, \widehat{\boldsymbol{\theta}}_c) = \max \left(0, 1 - \sum_{j \in \mathcal{Y}} q(y = j | \mathbf{x}, \widehat{\boldsymbol{\theta}}_j) \right) , \quad (3.7.0.15)$$

where $q(y = j | \mathbf{x}, \widehat{\boldsymbol{\theta}}_j) = \max \left(0, \widehat{\boldsymbol{\theta}}_j^T \boldsymbol{\phi}(\mathbf{x}) \right)$.

Algorithm 15 $LSAD(X, \rho, \sigma, B)$

Inputs: X - input data, ρ - regularization parameter,
 σ - kernel length space parameter, B - number of support points

Output: score function $s(\mathbf{x})$

- 1: choose randomly B points $\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_B} \in X$
 - 2: $\Phi = (\boldsymbol{\phi}(\mathbf{x}_1), \dots, \boldsymbol{\phi}(\mathbf{x}_n))^T$
 - 3: **for** $i \in \{1, \dots, c\}$ **do**
 - 4: $\mathbf{m}_i = (\delta_{y_1, i}, \dots, \delta_{y_n, i})^T$
 - 5: $\hat{\boldsymbol{\theta}}_i = (\Phi^T \Phi + \sigma \mathbf{I}_B)^{-1} \Phi \mathbf{m}_i$
 - 6: **end for**
 - 7: $\hat{\boldsymbol{\theta}}_* = \sum_{j \in \mathcal{Y}} \hat{\boldsymbol{\theta}}_j$
 - 8: $s(\mathbf{x}) = \max \left(0, 1 - \hat{\boldsymbol{\theta}}_*^T \boldsymbol{\phi}(\mathbf{x}) \right)$
 - 9: **return** $s(\mathbf{x})$
-

Theoretical Modelling Approach

As shown in the algorithm the final scoring function, which is defined as the estimated conditional probability of an outlier, is given by

$$s(\mathbf{x}) = \widehat{q}(y = * | \mathbf{x}, \hat{\boldsymbol{\theta}}_*) = \max \left(0, 1 - \hat{\boldsymbol{\theta}}_*^T \boldsymbol{\phi}(\mathbf{x}) \right) . \quad (3.7.0.16)$$

We recognize, that the training data gets modelled by $\hat{\boldsymbol{\theta}}_*^T \boldsymbol{\phi}(\mathbf{x})$, which is a linear model of the data point \mathbf{x} in the transformed space $\boldsymbol{\phi}(\mathbb{R}^p)$. Then the anomaly score is obtained by first computing one minus this linear model, which is again a linear model. Further, we cut off the estimate at zero, to guarantee the condition of positive probability estimates. Thus, we model the assignment function $\widehat{H}(\mathbf{x})$ by a linear model (in $\boldsymbol{\phi}(\mathbf{x})$), which is in addition cut off at zero. Of course, in the real space of the data point $\mathbf{x} \in \mathbb{R}^p$, the scoring function looks a lot more complicated, since the transformation function $\boldsymbol{\phi}(\mathbf{x})$ is highly non-linear.

The LSAD approach is related to density estimation through the equation (3.7.0.16), i.e. the (at zero truncated) hyperplane $\hat{\theta}_*^T \phi(\mathbf{x})$ in the transformed space $\phi(\mathbb{R}^p)$, can potentially be used to define a density estimation method.

Handling Different Datatypes

Since the method is very similar to the one-class SVM, the handling of categorical and ordinal variables happens in exactly the same way using the same arguments.

Choice of Parameters

- Kernel coefficient** $\sigma \in \mathbb{R}_{>0}$ (default: "automatic", chosen by the method):
 This parameter is the bandwidth parameter used for the Gaussian kernel and determines how complex the model is allowed to be. The smaller we choose this parameter the more complex is the local structure of the model allowed to be. Thus this parameter controls the smoothness of the model. Its definition and usage is very similar to the σ parameter in the OSVM. As default we let the algorithm choose this parameter by itself.
- Adaptation vs. Regularization trade-off parameter** $\rho \in \mathbb{R}_{>0}$ (default: $\rho = 0.6$):
 This parameter balances the trade-off between adaptation of the model to the data and the regularization term. The higher we choose this value, the more sensitive is the algorithm to outliers.
- Number of support points** $B \in \mathbb{N}$ (default: $B = 1000$):
 This parameter defines how many points are chosen to represent the data. If one increases this number the model captures the data in more detail. But a word of caution has to be said if one wants to increase this number. Since the parameter B determines the size of the Φ matrix ($B \times n$) which has to be stored during the training process, one can easily fill up the complete memory and force the algorithm to crash. Even if this memory overflow doesn't happen, the matrix multiplication in equation (3.7.0.7) can become a huge burden.

Advantages

- Because of the close functional relationship of LSAD with the OSVM method, the advantages are overall the same.
- The LSAD method is generally faster than the OSVM, because it can be trained without iterations, in a one step calculation due to the least squares error. But nevertheless it's still slow for too many samples.
- The method is very robust against noise features (see section 5.8).

Disadvantages

- The disadvantages are the same as for the OSVM due to the reasons mentioned above.

3.8 Probabilistic PCA

The idea of the Probabilistic PCA (PPCA) (Tipping and Bishop, 1999) method for anomaly detection is, to impose a generative probabilistic model on PCA. The likelihood under the model is further used, as a measure for anomaly/novelty of an new observation.

Anomaly Assumption

For the PPCA anomaly method we have to assume two things:

- *Low density assumption:* We assume that outliers occupy a low density region of the data space.
- *Modelling assumption:* We assume that the high density regions of the data can be captured well by the generative PPCA model.

Algorithm

The standard/traditional formulation of the PCA on p -dimensional data points $\mathbf{x}_1, \dots, \mathbf{x}_n$ is well known, but we will quickly state it here. Note that the notation in this section is not standard for PCA, but is chosen such that it matches with the notation in the rest of the thesis and especially with the notation of the rest of the chapter. The q -dimensional principal axes $\mathbf{l}_1, \dots, \mathbf{l}_q$ ($q \leq n$) are defined using orthonormal axes in the original data space. This axes are chosen in a way, that the retained variance under the projection on these axes is maximal.

It can be shown that this principal axes correspond to the q dominant eigenvectors. By the q dominant eigenvectors we mean the eigenvectors, that belong to the largest q associated eigenvalues of the covariance matrix.

In more detail, the sample covariance matrix of the data is given by

$$\mathbf{S} = \sum_i (\mathbf{x}_i - \bar{\mathbf{x}}) (\mathbf{x}_i - \bar{\mathbf{x}})^T / N, \quad (3.8.0.1)$$

where $\bar{\mathbf{x}}$ denotes the sample mean vector. Then the principal components are given by the eigenvectors that correspond to the first q biggest eigenvalues ($\lambda_1 > \dots > \lambda_q > \dots > \lambda_n$) of \mathbf{S} ,

$$\mathbf{S} \mathbf{l}_j = \lambda_j \mathbf{l}_j, \quad j = 1, \dots, q. \quad (3.8.0.2)$$

We define the q -dimensional reduced representation of the observed vector \mathbf{x}_i by

$$\mathbf{y}_i = \mathbf{L}^T (\mathbf{x}_i - \bar{\mathbf{x}}), \quad (3.8.0.3)$$

where $\mathbf{L} = (\mathbf{l}_1, \dots, \mathbf{l}_q)$. This map is the transformation function, that maps values in the original data space \mathbb{R}^p to the principal subspace \mathbb{R}^q .

Further it can be shown, that the projection onto the principal subspace minimizes the squared reconstruction error

$$\sum_i \|\mathbf{x}_i - \hat{\mathbf{x}}\|^2, \quad (3.8.0.4)$$

where the optimal linear reconstruction is given by

$$\hat{\mathbf{x}} = \mathbf{L}\mathbf{y}_i + \bar{\mathbf{x}}. \quad (3.8.0.5)$$

After the repetition of the standard PCA done so far, we can cover the specialities of the Probabilistic PCA. The basis of PPCA is a latent variable model. The basic idea of a generative model is to describe the d -dimensional observed data points \mathbf{x} as a function of q -dimensional latent variables \mathbf{l} ($q < p$) and (\mathbf{x} -independent) noise ϵ . We use the following formula to define the generative model

$$\mathbf{x} = \mathbf{f}(\mathbf{l}; \mathbf{w}) + \epsilon, \quad (3.8.0.6)$$

where $\mathbf{f}(\cdot; \cdot)$ is the function that defines the relation between the observed data \mathbf{x} and the latent variables \mathbf{l} . Thus the equation (3.8.0.6) describes a distribution over the data space which can be used to generate data by simply sampling from the distributions of \mathbf{x} and ϵ and the evaluating of the right hand side of the equation.

For the PPCA model we assume the following distributions for noise and latent variables

$$\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}), \quad (3.8.0.7)$$

$$\mathbf{l} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \quad (3.8.0.8)$$

Further the mapping \mathbf{f} is defined as

$$\mathbf{f}(\mathbf{l}, \mathbf{W}) = \mathbf{W}\mathbf{l} + \boldsymbol{\mu}, \quad (3.8.0.9)$$

where $\boldsymbol{\mu}$ permits the data to have non-zero mean. Thus one can show that the distribution of the observed data is $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$, where $\mathbf{C} = \sigma^2 \mathbf{I} + \mathbf{W}\mathbf{W}^T$

Using this knowledge the log-likelihood of the data can be written as

$$\mathcal{L} = \sum_{i=1}^N \ln(p(\mathbf{x}_i | \mathbf{W}, \boldsymbol{\mu}, \sigma^2)) = -\frac{N}{2} \left(p \ln(2\pi) + \ln|\mathbf{C}| + \text{tr}(\mathbf{C}^{-1} \mathbf{S}) \right), \quad (3.8.0.10)$$

where

$$\mathbf{S} = \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T, \quad (3.8.0.11)$$

the sample covariance matrix of the observed data $\mathbf{x}_1, \dots, \mathbf{x}_n$.

We define the anomaly score of the PPCA as the estimated log-likelihood of a sample under the generative model described before:

$$s(\mathbf{x}) = -\frac{p}{2} \ln(2\pi) - \frac{1}{2} \ln|\hat{\mathbf{C}}| - \frac{1}{2} (\mathbf{x} - \hat{\boldsymbol{\mu}})^T \hat{\mathbf{C}}^{-1} (\mathbf{x} - \hat{\boldsymbol{\mu}}) \quad (3.8.0.12)$$

To estimate the parameters of the model we use maximum likelihood estimation. Computing the MLE solutions of the parameters with taking derivatives we arrive at estimators.

The MLE of the parameter $\boldsymbol{\mu}$ is simply the mean of the data points

$$\hat{\boldsymbol{\mu}} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i. \quad (3.8.0.13)$$

For MLE of σ^2 we first have to compute an eigendecomposition of the matrix \mathbf{S} . Let \mathbf{U}_q be the $d \times q$ matrix of q (column) eigenvectors of \mathbf{S} and $\mathbf{\Lambda}_q$ the $q \times q$ diagonal matrix that contains the corresponding eigenvalues $\lambda_1, \dots, \lambda_q$. Then the eigenvalue decomposition $\mathbf{S} = \mathbf{U}_q^T \mathbf{\Lambda}_q \mathbf{U}_q$ holds.

The MLE estimate of σ^2 is given by

$$\widehat{\sigma^2} = \frac{1}{p-q} \sum_{j=q+1}^p \lambda_j. \quad (3.8.0.14)$$

This parameter has the interpretation of the average variance that is "lost" per dimension that is thrown away by the PPCA model.

The next step is to estimate the weight matrix \mathbf{W} ,

$$\widehat{\mathbf{W}} = \mathbf{U}_q \left(\mathbf{\Lambda}_q - \widehat{\sigma^2} \mathbf{I} \right)^{1/2} \mathbf{R}, \quad (3.8.0.15)$$

where \mathbf{R} is an arbitrary $q \times q$ orthogonal rotation matrix.

Using this we can estimate \mathbf{C} using

$$\widehat{\mathbf{C}} = \widehat{\sigma^2} \mathbf{I} + \widehat{\mathbf{W}} \widehat{\mathbf{W}}^T. \quad (3.8.0.16)$$

Algorithm 16 *PPCA(X, q)*

Inputs: X - input data, q - subspace dimension

Output: score function $s(\mathbf{x})$

1: Compute

$$\widehat{\boldsymbol{\mu}} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i, \quad \mathbf{S} = \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \widehat{\boldsymbol{\mu}})(\mathbf{x}_i - \widehat{\boldsymbol{\mu}})^T. \quad (3.8.0.17)$$

2: Compute eigenvalue decomposition $(\mathbf{U}_q, \mathbf{\Lambda}_q)$ of \mathbf{S} .

3: Compute

$$\widehat{\sigma^2} = \frac{1}{p-q} \sum_{j=q+1}^p \lambda_j, \quad \widehat{\mathbf{W}} = \mathbf{U}_q \left(\mathbf{\Lambda}_q - \widehat{\sigma^2} \mathbf{I} \right)^{1/2} \mathbf{R}, \quad \widehat{\mathbf{C}} = \widehat{\sigma^2} \mathbf{I} + \widehat{\mathbf{W}} \widehat{\mathbf{W}}^T \quad (3.8.0.18)$$

4: $s(\mathbf{x}) = -\frac{p}{2} \ln(2\pi) - \frac{1}{2} \ln |\widehat{\mathbf{C}}| - \frac{1}{2} (\mathbf{x} - \widehat{\boldsymbol{\mu}})^T \widehat{\mathbf{C}}^{-1} (\mathbf{x} - \widehat{\boldsymbol{\mu}})$

5: **return** $s(\mathbf{x})$

Note that the implementation we use in the experiments section (section 5) uses the SVD decomposition instead of the eigenvalue decomposition because of computational complexity reasons.

Theoretical Modelling Approach

The scoring function, is nothing else than the log-transformed multivariate normal density function with parameters $\widehat{\boldsymbol{\mu}}$ and $\widehat{\mathbf{C}}$

$$s(\mathbf{x}) = \log(f(\mathbf{x}|\widehat{\boldsymbol{\mu}}, \widehat{\mathbf{C}})) = -\frac{p}{2} \ln(2\pi) - \frac{1}{2} \ln |\widehat{\mathbf{C}}| - \frac{1}{2} (\mathbf{x} - \widehat{\boldsymbol{\mu}})^T \widehat{\mathbf{C}}^{-1} (\mathbf{x} - \widehat{\boldsymbol{\mu}}). \quad (3.8.0.19)$$

Thus the algorithm assumes that the assignment function can be estimated using a (single) Gaussian probability density $\hat{H}(\mathbf{x}) = g(f(\mathbf{x}|\hat{\boldsymbol{\mu}}, \hat{\mathbf{C}}))$, where g is an appropriate monotone function.

The relation of the PPCA method to density estimation is straight forward through this Gaussian density function. We can compute this density function directly using the computed quantities $\hat{\boldsymbol{\mu}}$ and $\hat{\mathbf{C}}$.

Handling Different Datatypes

The handling of categorical and ordinal data is done again with dummy encoding. Nevertheless the algorithm often improves its performance by a huge margin for mixed type data sets if one uses data representation.

Choice of Parameters

- **Subspace dimension** $q \in \{1, \dots, p\}$ (default: $q = p$):
This parameter specifies how many dimension the PCA subspace should have. Since we want to build a model of the training data which is as accurate as possible, $q = p$ is a descent default choice.

Advantages

- The idea of the PCA method is conceptionally very simple.
- The method (in particular the training process) is often very fast, since the method mainly consists of computing the eigenvalue or singular value decomposition.
- Using some form of representation before applying the PCA anomaly detection method often results in comparatively good results. Further one has to remove features with almost zero variance before applying the method for the same reasons.

Disadvantages

- The scoring function has the form of a multivariate Gaussian density function, which is very restricting since it allows for only one cluster. Further the assumption of normal distributed data is likely not to fit well to the data. This is especially the case for dummy encoded categorical features. If this assumption doesn't hold it can (but does not have to) result in very poor scoring functions.
- The method doesn't work well on a too small number of samples, i.e. one shouldn't use it below a number of 100 samples since the algorithm gets into numerical problems.

3.9 Gaussian Mixture Model

The idea of the GMM (Bishop (2007), chapter 9) is to estimate the distribution of the data by a mixture of Gaussian densities. Then the density function of the mixture model is used as an anomaly score, i.e. the smaller the density the bigger the anomaly score.

Anomaly Assumption

For the GMM anomaly method we have to assume two things:

- *Low density assumption*: We assume that outliers occupy a low density region of the data space.
- *Modelling assumption*: We assume that the high density regions of the data can be captured by the Gaussian mixture model.

Algorithm

We assume that the density of the data is a mixture of k Gaussian modes given by densities $g_{\theta_j}(\mathbf{x}) = g(\mathbf{x}|\theta_j)$, $j = 1, \dots, k$ of $\mathcal{N}(\boldsymbol{\mu}_j, \Sigma_j)$, with parameters $\theta_j = (\boldsymbol{\mu}_j, \Sigma_j)$. Additionally we assume isotropic Gaussian mixture components, i.e. the covariance matrix is diagonal with constant variances $\Sigma_j = \sigma_j^2 \mathbf{I}_p$ (circular symmetric).

Assume that the different densities have weights w_1, \dots, w_k . Thus the mixture density

$$f_{\theta}(\mathbf{x}) = \sum_{j=1}^k w_j g_j(\mathbf{x}, \theta_j), \quad \sum_{j=1}^k w_j = 1, \quad (3.9.0.1)$$

where the parameter $\theta = (w_1, \dots, w_k; \boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k; \Sigma_1, \dots, \Sigma_k)$ has to be estimated.

Before we can define the optimization algorithm, we first have to introduce some notation. We introduce the indicator matrix \mathbf{Z} containing the assignment variables $Z_{i,j} \in \{0, 1\}$ that indicate the membership of datapoint \mathbf{x}_i , i.e.

$$Z_{i,j} = \begin{cases} 1, & \text{if } \mathbf{x}_i \text{ generated by mode } g_j \\ 0, & \text{else} \end{cases}, \quad (3.9.0.2)$$

where $\sum_{j=1}^k Z_{i,j} = 1$, for all $i = 1, \dots, n$. We estimate this assignment variable by

$$\hat{Z}_{i,j} = \gamma_{i,j} = E[Z_{i,j} | \theta, \mathbf{x}_i] = P[Z_{i,j} = 1 | \theta, \mathbf{x}_i], \quad (3.9.0.3)$$

where Γ is the matrix containing the $\gamma_{i,j}$ values equivalent to the \mathbf{Z} matrix.

The estimation of this parameter happens with the EM-algorithm, which consists of two iterating steps, an "expectation" and a "maximization" step:

- *Expectation-Step*: In the expectation step we estimate the conditional probability of the data point to belong to each mode given the current density parameter θ_{t-1} :

$$\gamma_{i,j}^{(t)} = E[Z_{i,j} | \theta^{(t-1)}, \mathbf{x}_i]. \quad (3.9.0.4)$$

- *Maximization-Step*: In the maximization step we estimate the new parameter θ_t such that they maximize the log-likelihood of the assignment variables $\gamma_{i,j}$

$$\theta^{(t)} = \arg \max_{\theta} L(\theta, X, \Gamma^{(t)}), \quad (3.9.0.5)$$

where $L(\theta, X, \Gamma) = \prod_{i=1}^n \prod_{j=1}^k (w_j g_{\theta_j}(\mathbf{x}_i))^{\gamma_{i,j}}$ is the likelihood function of the data.

The initialization of the EM-algorithm can either be done at the expectation step by choosing initial parameters θ or at the maximization step by choosing the initial assignment of the variables according to an initial clustering (eg. initial k -means clustering).

Rewriting the two generally formulated equations (3.9.0.4), (3.9.0.5) of the ER steps with our Gaussian mixture model assumptions, we end up with the following algorithm:

Algorithm 17 *GMM*(X, k)

Inputs: X - input data, k - number of clusters

Output: score function $s(\mathbf{x})$

- 1: **repeat** for $t = 1, 2, \dots$
- 2: Calculate for all i, j // E-step

$$\gamma_{i,j}^{(t)} = \frac{w_j^{(t)} g_{\theta_j^{(t)}}(\mathbf{x}_i)}{f_{\theta^{(t)}}(\mathbf{x}_i)} \quad (3.9.0.6)$$

- 3: Calculate for all j // M-step

$$\mu_j^{(t+1)} = \frac{\sum_{i=1}^n \gamma_{i,j}^{(t)} \mathbf{x}_i}{\sum_{i=1}^n \gamma_{i,j}^{(t)}} \quad (3.9.0.7)$$

$$(\sigma_j^2)^{(t+1)} = \frac{\sum_{i=1}^n \gamma_{i,j}^{(t)} (\mathbf{x}_i - \mu_j^{(t+1)})^2}{\sum_{i=1}^n \gamma_{i,j}^{(t)}} \quad (3.9.0.8)$$

$$w_j^{(t+1)} = \frac{1}{n} \sum_{i=1}^n \gamma_{i,j}^{(t)} \quad (3.9.0.9)$$

- 4: **until** changes small enough
 - 5: $s(\mathbf{x}) = -f_{\theta^{(t)}}(\mathbf{x})$
 - 6: **return** $s(\mathbf{x})$
-

Theoretical Modelling Approach

The scoring function is defined as the negative density of the mixture of Gaussian modes

$$s(\mathbf{x}) = -f_{\theta}(\mathbf{x}) = 1 - \sum_{j=1}^k w_j g_{\theta_j}(\mathbf{x}). \quad (3.9.0.10)$$

Thus the scoring function consists of a weighted sum of Gaussian modes that are scaled by weights w_j which indicate the likelihood that a sample is drawn from cluster j . Thus our estimate of the assignment function $\hat{H}(\mathbf{x})$ is given by the negative probability density.

Therefore, the GMM is obviously related to density estimation through the explicit use of the density f_{θ} in the definition of the scoring function above.

Handling Different Datatypes

The handling of categorical and ordinal data is done again with dummy encoding. Of course this deviates considerably from the assumption that the data is generated by a mixture of Gaussian modes but still produces reasonable scores for some datasets. However the GMM usually performs better on (numerical) encoded data.

Choice of Parameters

- **Number of clusters** $k \in \mathbb{N} \setminus \{0\}$ (default $k = 10$):

The number of clusters is obviously the most important parameter for the performance of the algorithm. Similar to the k -means algorithms one can argue that using enough clusters will allow to capture more structure in the data and bigger clusters may be represented by multiple modes. But compared to k -means one has to be cautious with thoughtless increasing of the number of clusters since the algorithm runtime considerably increases. In addition the GMM clusters have a lot more flexibility (compared to k -means). This is mainly caused by the $\frac{p(p+1)}{2}$ parameters of the covariance matrix, e.g. this amounts to a number of 55 covariance parameters for $p = 10$ feature dimensions. Thus tests show that $k = 10$ is in general a good default value.

Advantages

- The idea is conceptionally very simple.
- The method allows for a lot of flexibility with a low number of clusters k because the covariance matrix has a lot of parameters that can adapt the shape of the Gaussian modes to the data.
- The method shows a robust behaviour against noise features (see section 5.8).

Disadvantages

- It is well known that the EM algorithm can be quite slow and needs a lot of iterations before it converges.
- The method is slow for big datasets.
- The method relies on the assumption that the data can be modelled well by a mixture of Gaussian modes. If this assumption holds true approximately, the model works well, but otherwise it can have a poor performance.

Chapter 4

Representation Models

When investigating the usefulness of different anomaly detection algorithms on different types of datasets/problems, one should think as well how the data is represented to the algorithms. Our insight into work done so far for anomaly detection, shows that there hasn't been done much research investigating that idea.

It is well known that manual feature engineering is an important step in many machine learning pipelines and often involves a lot of time and thought. Thus it is interesting to think about ways to automate this step, in order to save time or even to come up with features that no human expert would come up with. The choice of features even becomes more critical in the context of anomaly detection compared to supervised learning (classification or regression). The reason for this is, that the anomaly detection algorithms cannot decide which features comprise information and should be used for training and which ones are just noise and thus should be disregarded for training.

Representation learning tries to solve the potential problems of feature engineering to extract useful and discriminative information from the data and consequently decrease the dependence on feature engineering step. The algorithms need to learn how to identify and disentangle explanatory factors, which are often hidden in low-level distributed data with small signal-ratio each individual feature.

Thus, an often used way to define representation learning is to mention what it eventually will be used for. A useful representation is one that works well as an input to the next step in the machine learning pipeline which could be a supervised learning method or as in our case an anomaly detection method.

This chapter is largely formed by information from the Representation Learning review by [Chandola et al. \(2009\)](#), which gives a detailed overview over the different representation approaches and their properties.

4.1 Principal Component Analysis

Probably the most frequently used way to represent data is to use the first $q < p$ Principal Components computed by Principal Component Analysis (Bishop (2007), chapter 12). It has already been used for a long time to extract features from data as a preprocessing step to supervised learning algorithms.

The main idea of PCA is given a (entered) data matrix \tilde{X} ($n \times p$) one computes a rotation \mathbf{L} ($n \times q$) of the centred data such that:

- The first column of \mathbf{L} has largest possible sample variance,
- The second column has largest possible sample variance and is perpendicular/uncorrelated to the first one,
- The third column has largest possible sample variance and is perpendicular/uncorrelated to the first two columns,
- ...
- The q -th column has largest possible sample variance and is perpendicular/uncorrelated to the first $q - 1$ columns.

To obtain the PCA represented data we use the rotation matrix \mathbf{L} to project data points into a linear subspace, which keeps as much variance as possible.

Representation Idea

The idea behind the PCA representation is to choose the linear subspace (hyperplane) of the centred data, that preserves as much variance in the data as possible and project the data on it. Thus, PCA is generally good in representing a dataset, in which the data points lie in a lower dimensional hyperplane of the original data.

Anomaly Assumption

When using the PCA representation for anomaly detection, the data needs to satisfy, that the signal, which determines whether a data point is an anomaly or not, lies in the high variance directions of the data. Hence in this case the information, that distinguishes the anomalies from normal points, will be preserved by this representation method. An example for both of these cases, the successfully and unsuccessful preservation of the anomaly signal, can be seen in Figure 4.1.

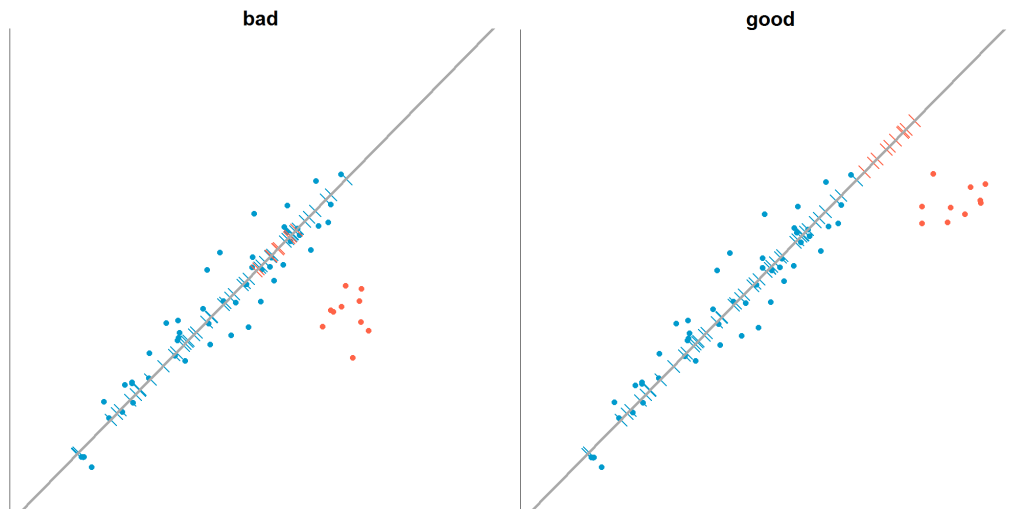


Figure 4.1: This two plots show two examples of train (blue) and test (red) data sets and their representation in the PCA subspace (blue and red ticks). The blue dataset, which is identical in both examples, has been used to fit the PCA model (grey). In both plots we project the dataset onto the first principal component. In the example on the right hand side, we can see that the anomaly signal lies in the high variance direction of the training data and thus can be preserved by PCA and vice versa for the example at the left hand side.

Algorithm

Since the idea and the computational steps are already discussed in the chapter of the PPCA model (section 3.8) we will only cover details of the PCA dimension reduction here.

The equation to transform the data is then given by the equation (3.8.0.3) in the section 3.8 about the Probabilistic PCA method and states

$$\mathbf{y}_i = \mathbf{L}^T (\mathbf{x}_i - \bar{\mathbf{x}}) , \quad (4.1.0.1)$$

with the $\mathbf{L} = (\mathbf{l}_1, \dots, \mathbf{l}_q)$ the matrix containing the eigenvectors of the covariance matrix \mathbf{S} .

Thus the algorithm to compute the transformation function $\mathbf{t}(\mathbf{x})$ computes first the mean vector of the data $\bar{\mathbf{x}}$, then computes the covariance matrix \mathbf{S} and performs eigenvalue (or SVD) decomposition of this matrix. With this quantities computed, the transformation function can be written down directly by plugging-in the mean vector $\bar{\mathbf{x}}$ and rotation matrix \mathbf{L}_q into the transformation formula.

We recognize that the transformation of a new data point \mathbf{x} just consist of the matrix multiplication of the rotation matrix \mathbf{L} with the centred new datapoint $\mathbf{x} - \bar{\mathbf{x}}$.

Algorithm 18 $PCA(X, q)$

Inputs: X - input data, q - subspace dimension ($q \leq p$)**Output:** transformation function $\mathbf{t}(\mathbf{x}) : \mathbb{R}^p \rightarrow \mathbb{R}^q$

1: Compute

$$\mathbf{S} = \frac{1}{N} \sum_i (\mathbf{x}_i - \bar{\mathbf{x}}) (\mathbf{x}_i - \bar{\mathbf{x}})^T . \quad (4.1.0.2)$$

2: Compute eigenvalue decomposition $(\mathbf{L}_q, \mathbf{\Lambda}_q)$ of \mathbf{S} , st. $\mathbf{S} = \mathbf{L}_q \mathbf{\Lambda}_q \mathbf{L}_q^T$.3: $\mathbf{t}(\mathbf{x}) = \mathbf{L}_q^T (\mathbf{x} - \bar{\mathbf{x}})$ 4: **return** $\mathbf{t}(\mathbf{x})$

Choice of Parameters

- **Compression factor** $\rho \in [0, 1]$ (default: $\rho = 0.8$):

The compression factor is the most important parameter for determining the usefulness of the representation. We will closer investigate the effect of the compression factor on the performance of the anomaly detection algorithms in chapter 5.

Advantages

- The method is conceptually very simple. This potentially allows for interpretation, e.g. one can investigate the loadings of the principal components to understand what they are composed of.
- The only parameter to choose is the compression factor ρ . This can be very handy, because every parameter that needs to be specified, makes the method more difficult to control.

Disadvantages

- The method is restricted to a projection into a linear subspace of the original data. Thus the method may fail to capture complex, non-linear structure of the data.
- The method is designed to preserve as much variance of the data as possible. Thus it may end up with an unusable representation, if the structure we want to preserve lies in a low variance direction of the data.

4.2 Autoencoder

The usage of an Autoencoder (AE) for the task of representation learning is quite common as well and very convenient. One can show, that an Autoencoder with linear activation function results in a PCA (see (Chandola et al., 2009)). Therefore one can interpret an Autoencoder with a non-linear activation function as a non-linear generalization of PCA.

Since we use exactly the same AE as described in the corresponding section of the anomaly detection methods (section 3.4), we refer to this section for the description and algorithm. In this section, we only describe the aspects concerning its usage as a representation model.

Representation Idea

The usage of a non-linear activation function and multiple layers between the bottleneck-layer (or representation-layer (RL)) and the input layer allow for a lot more flexible and abstract way to encode our dataset, as for example a PCA. The bigger the number of layers between the RL and the input layer or output layer respectively, the more abstract the representation. Further the number of units in the RL specify how much "compression" happens in the representation step with respect to the original data.

Anomaly Assumption

Similar to the PCA, which embeds the data on a lower dimensional hyperplane, the AE embeds the data in a lower dimensional manifold of the original data.¹ This implies that the representation works well in the anomaly detection scenario if the normal-/anomaly-signal doesn't lie in directions perpendicular to this manifold. An example which illustrates this fact is given in Figure 4.2.

Algorithm

We use the same Autoencoder as in section 3.4 and thus the algorithm is exactly the same as described there. Further we will use an undercomplete Autoencoder to restrict the capacity, which makes sense because we are interested to keep the representation dimension as small as possible.

¹For details, see chapter 14.2 of the book Goodfellow et al., 2016.

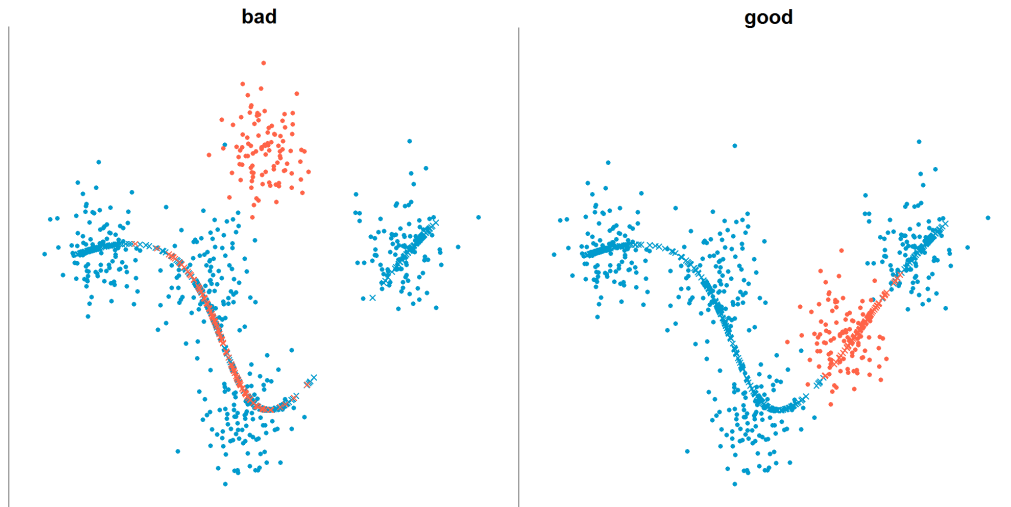


Figure 4.2: This two plots show two examples of train (blue) and test (red) data sets and their reconstruction of the AE function (only one unit in bottleneck-layer) which corresponds to the representation of the Autoencoder (blue and red crosses). Because the data is four dimensional, the two plots do not show the original data, but a two dimensional projection of it. We see that in the case where the anomaly signal lies in a direction perpendicular to the manifold, the signal can't be preserved (left hand side). But in the case where the signal lies in a direction parallel to the manifold, it can be preserved (right hand side).

Choice of Parameters

- Compression factor** $\rho \in [0, 1]$ (default: $\rho = 0.6$):
 The compression factor is the most important parameter for determining the performance of the representation. We will closer investigate the effect of the compression factor on the performance of the anomaly detection algorithms in chapter 5.
- Hidden-layer architecture** (default: $(\lfloor p \cdot \rho \rfloor, \lfloor p \cdot \rho \rfloor, \lfloor p \cdot \rho \rfloor)$):
 In principle there are a lot of different ways to choose the hidden-layer architecture. Testing around showed that the following three layer architecture $(\lfloor p \cdot \rho \rfloor, \lfloor p \cdot \rho \rfloor, \lfloor p \cdot \rho \rfloor)$, gives good results. This list of integers $\lfloor p \cdot \rho \rfloor$ means that the AE has three hidden layers, each having this number of neurons in it.
- Number of training epochs** (default: 1.0):
 Setting the number of epochs equals to 1.0 produces good results.
- Activation function** (default: *tanh*):
 This parameter describes which activation function $\varphi(\cdot)$ we use in the neurons. Testing different activation functions showed that *tanh* delivered the best results.

Advantages

- Due to the multiple layer architecture and the ability of neural networks to deal with complicated (non-linear) interactions, the Autoencoder is well suited to represent complex data.
- The AE can deal well with categorical features by interpreting them as dummy variables (see section 3.4). Therefore the AE is well suited to represent mixed-type data.

Disadvantages

- There are many parameters to choose (e.g. hidden-architecture, number of training epochs) for the AE. Unfortunately, their optimal choice is not clear in a general situation and can vary from case to case. Thus one has to choose the parameters in an ad-hoc way.
- An interpretation of the encoded data is almost impossible, because a non-linear transformation of a weighted sum of other activations in the network is not easy to interpret.

4.3 Entity Embedding

Entity Embedding (EE) has recently been used successfully in a Kaggle competition and has been discussed further in the paper of [Guo and Berkhahn \(2016\)](#). The idea of the EE method is to perform a "supervised learning-motivated" embedding of categorical (and numerical) features using a neural net. The authors used a multiple-layer neural network trained to solve a regression problem as an embedding method. They trained the network and propagated the data until the first hidden layer. Then they used this as their embedded representation and supplied it as input to other regression methods.

However in the anomaly detection scenario we lack labeled data to train a neural network on and thus have to engineer a dataset with normal values and anomalies in order to apply EE. We choose the idea of resampling a synthetic anomaly class, stated in the section about Unsupervised Random Forest (section 3.2), to produce such a labeled dataset. We use the exact same sampling idea as used in the URF method, train a NN on this data and use the representation in the first layer as a representation of the original data (see Figure 4.3).

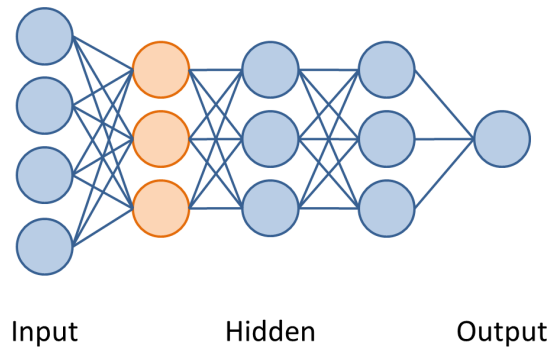


Figure 4.3: This figure shows a schematic picture of an Entity Embedding neural network, which learns to predict if a datapoint belongs to the original or the synthetically generated dataset. The the activations of a datapoint in the first layer (orange) gets extracted and used as the Entity Embedding representation for the original datapoint.

Representation Idea

The idea of the EE representation is that a deep neural network that is trained to distinguish the real data from a synthetically generated anomaly dataset (representing the future anomalies). Then the first layer of its architecture is used to encode the data in from that is useful to distinguish anomalies from normal data. In further layers of the neural network, this representation is then used to solve the actual classification task.

Thus the EE idea is basically to do a supervised approach to represent the data, because we use the internally generated representation of a supervised learner as a feature representation.

Anomaly Assumption

For the method to work well, we have to assume the following two things:

- *Synthetic samples contain anomaly class*: The first assumption deals with the synthetic generated dataset that is used as a substitution for the anomaly class to train the neural network. The distribution of the synthetic dataset has to satisfy the assumption that its probability is bigger than the probability of the original dataset in regions where the normal distribution is more probable than the anomaly data set. In formal terms this means

$$f_F(\mathbf{x}) < f_G(\mathbf{x}) \iff f_F(\mathbf{x}) < f_{G'}(\mathbf{x}), \quad (4.3.0.1)$$

where G' is the distribution function of the synthetically generated anomaly dataset and F and G the distribution functions of the normal and anomaly distribution respectively (see section 2.6).

- *NN can learn the classification task*: The second assumption is that the neural network can appropriately learn the classification task. If the NN doesn't satisfy this assumption the first layer is not very likely to give a useful representation of the data for anomaly detection.

Algorithm

We use a common deep neural network, which has the same structural properties as the Autoencoder in section 3.4. The only difference are the hidden layer architecture and the output, which is the target that should be predicted by the network. The target is in this case the label of the datapoints, i.e. if they belong to the real dataset or to the synthetically generated dataset. The tricks used to fit a neural network are already mentioned in the section 3.4 and thus we refer to that section for the actual algorithm.

Choice of Parameters

- **Compression factor** $\rho \in [0, 1]$ (default: $\rho = 1.0$):
The compression factor is the most important parameter for determining the performance of the representation. We will closer investigate the effect of the compression factor on the performance of the anomaly detection algorithms in chapter 5.
- **Hidden-layer architecture** (default: $(\lfloor p \cdot \rho \rfloor, \lfloor p \cdot \rho \rfloor, \lfloor p \cdot \rho \rfloor, \lfloor p \cdot \rho \rfloor)$):
In principle there are a lot of different ways to choose the hidden-layer architecture. Testing around showed that the following architecture $(\lfloor p \cdot \rho \rfloor, \lfloor p \cdot \rho \rfloor, \lfloor p \cdot \rho \rfloor, \lfloor p \cdot \rho \rfloor)$, gives good results. This list of integers $\lfloor p \cdot \rho \rfloor$ means that the AE has three hidden layers, each having this number of neurons in it.
- **Number of training epochs** (default: 1.0):
Setting the number of epochs equals to 1.0 produces good results.
- **Activation function** (default: *tanh*):
This parameter describes which activation function $\varphi(\cdot)$ we use in the neurons. Testing different activation functions showed that *tanh* delivered the best results.

Advantages

- We have an indirect control over the algorithm through the reference set sampling algorithm. This allows us to control what kind of anomalies the encoding is trained to distinguish from the real data. We can for example use the URF algorithm to find what parameters are useful to generate a reasonable anomaly set and use this knowledge to define the sampling procedure of the EE representation model.
- The neural network can encode complex non-linear structures and handles categorical variables nicely with dummy encoding.

Disadvantages

- If one of the anomaly assumptions is not fulfilled, i.e. either the neural network can't learn the classification task or the anomaly set does not represent the real anomaly set, the generated representations may be poor.
- There are a lot of parameters to choose.

Chapter 5

Experiments

In this chapter, we test the anomaly detection algorithms described in chapter 3 on different datasets, using the different data representations covered in chapter 4. We want to test how the different algorithms perform with respect to different number of training samples, representations and additional noise features.

In addition we want to visualize the distribution of the anomaly score sizes in a two dimensional toy dataset.

The goal is to compare, how the different algorithms perform in different settings. To do that, we need to compute the performances of the different scores produced by the different methods. Unfortunately the results are noisy in the sense, that they deviate from each other, depending on the split of the data in training and test set. Thus we repeat each setting for 5 different splitting seeds, take the mean of the different runs and plot the error bars to capture the variation in the results.

If nothing else is stated, all the experiments are done using the default parameters of the anomaly detection and representation methods, as stated in the corresponding "Choice of Parameters"-sections.

5.1 Datasets

To test our anomaly detection and representation methods, we use datasets from different application domains, with different compositions of data types and number of features. This section shortly mentions what the datasets are about and which properties they have.

Forest Cover Type

The Forest Cover Type dataset contains samples from 30×30 meter cells of different cartographic regions with specific cover types and corresponding features with measures like elevation, aspect, slope, distances to next water source, shade measures, latest wildfire information, wilderness area or soil type. It is worth mentioning that all measurements are cartographic variables only, i.e. the dataset contains no remotely sensed data. The data as well as the description of the dataset stem from the UCI Machine Learning Repository website ([Lichman, 2013](#)).

The "Covertype" feature is the label we will use to define the train/test dataset split. It describes the covertype of the cells (eg. Spruce/Fir, Lodgepole Pine, Aspen or Krummholz). We assume that the distribution of the different covertype classes differ in distribution with respect to the other ones. As anomaly class we choose "Type7" which corresponds to Krummholz.

For anomaly detection tests with this dataset we use a test set size of 50 000 samples and an anomaly ratio of 0.1.



Figure 5.1: This figure shows a t-SNE embedding of the Forest Cover Type dataset with $n = 7\,000$ samples (1 000 samples for each of the 7 levels).

One recognizes that the different label classes tend to occur with different frequencies in different area. However there is no clear separation of the levels. Thus we can expect the anomaly detection task to be quite challenging. When looking at the tests, we see that this holds true, with the best models performing around 0.85.

Credit Card Fraud

The dataset contains 284 807 credit card transactions made in two days during September 2013. Out of this dataset, 492 transactions are defined fraudulent and the rest is defined normal.¹

Due to confidential issues the data contains only the feature "Amount" and 28 principal components of a dataset for which no additional information is provided. The feature "Amount" is the amount of the transaction.

The feature which we use for splitting the data into train/test set is "Class", which has the levels "Class0" that indicates non-fraud samples and "Class1" which indicates fraud samples.

For anomaly detection tests with this dataset we use a test set size of 10 330 samples and an anomaly ratio of 0.05.

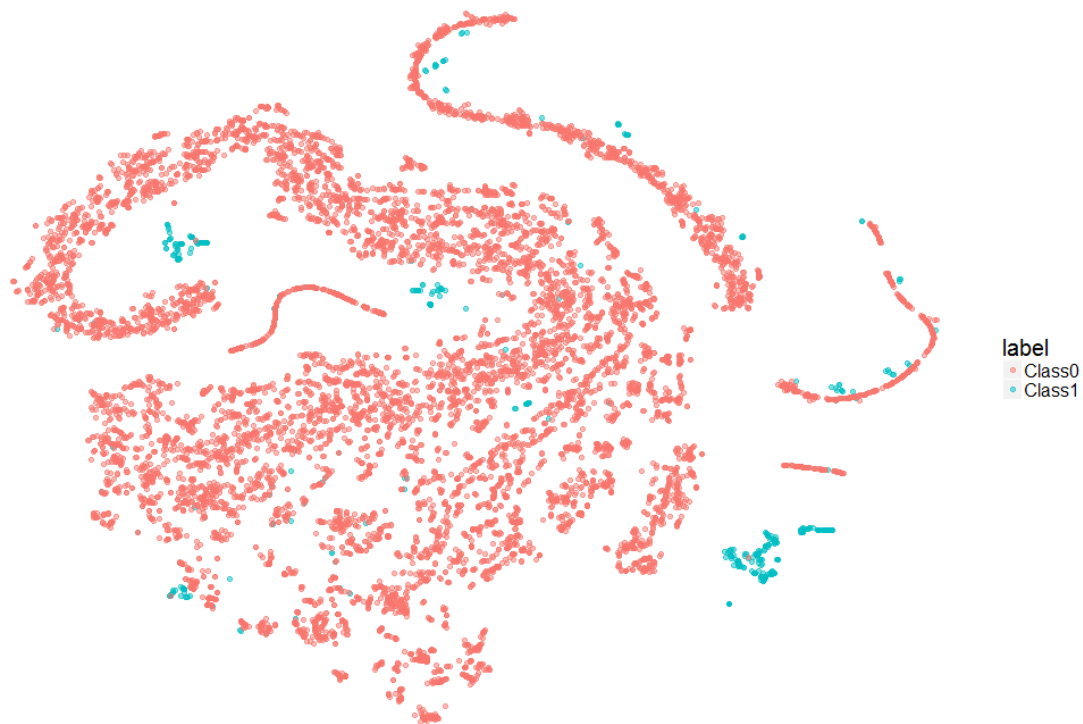


Figure 5.2: This figure shows a t-SNE embedding of the Credit Card Fraud dataset with $n = 10\,000$ samples (492 Class1, 9 508 Class0).

As we can see in Figure 5.2, the two classes are well separated and thus we should be able to achieve a reasonable detection performance in the anomaly detection task. This is confirmed by the results, where the best AUC performances achieved are around 0.95.

¹This dataset as well as the description are provided by the Kaggle homepage: <https://www.kaggle.com/dalpozz/creditcardfraud>.

Financial Data

The financial dataset information stems from the "Principles and Practice of Knowledge Discovery in Databases"-Challenge from the year 1999 (PKDD'99 Discovery Challenge).² The dataset has been constructed using a relational database containing 8 tables with information about transactions, accounts, clients, client demographics, etc.

In order to test that, we constructed a dataset of the transactions with 1 056 320 entries and 9 features, whereof 5 are categorical features. The features contain temporal information (day, month, year), type of the transaction (withdrawal, credit), the paid amount and balance, etc.

As label column we use the feature "k_symbol", which contains information about the nature of the transaction with levels such as insurance payment, payment for statement, interest credited, sanction interest if negative balance, household, old-age pension or loan payment. We use the level "loan payment" as the anomaly level, because we assume it to have another distribution then the other levels. For anomaly detection tests with this dataset we use a test set size of 10 000 samples and an anomaly ratio of 0.1.

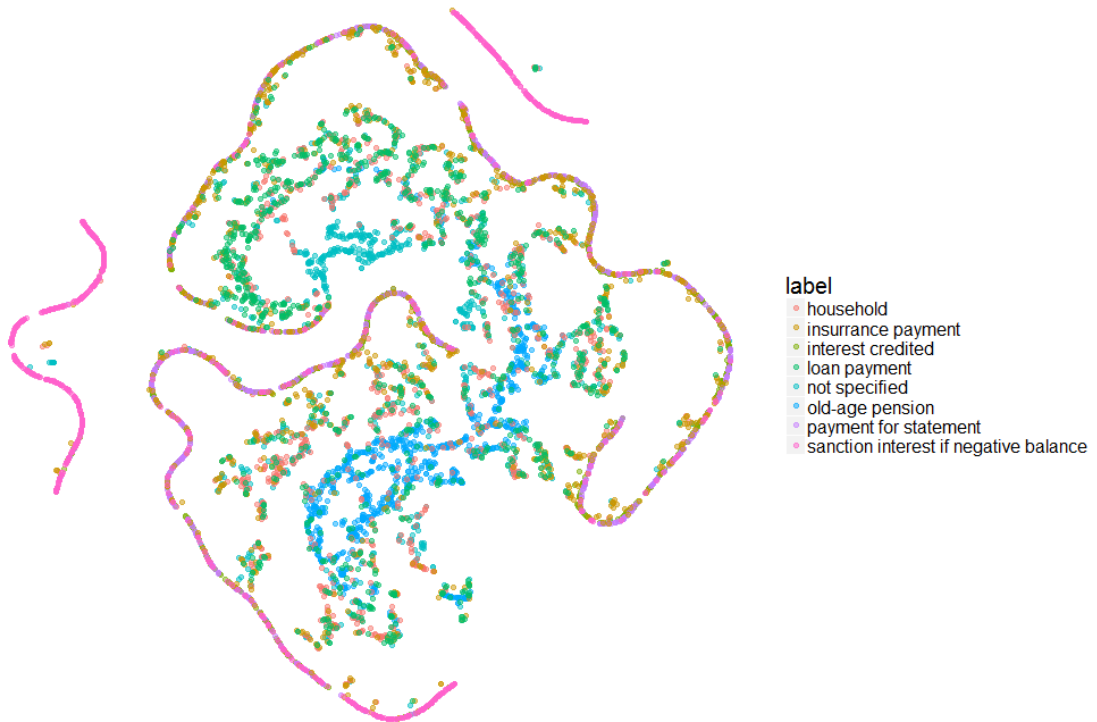


Figure 5.3: This figure shows a t-SNE embedding of the Credit Card Fraud dataset with $n = 8\,000$ samples (1 000 samples for each of the 8 levels).

The t-SNE plot shows that the loan payment class occurs in certain regions of the plot, but is always mixed up with other levels. Thus we expect a moderate performance. This is confirmed by the test, in which the best methods achieve AUC performances around 0.8.

²Description and dataset can be found at <http://lisp.vse.cz/pkdd99/berka.htm>.

Network Intrusion

The KDD Cup 1999 dataset contains 581012 data points about connections in a computer network.³ The goal is to distinguish "good" from "bad" connections which indicate network intrusions or attacks.

The 54 features consist of three main types: features that are based on the TCP connection, content features within a connection that are suggested by domain knowledge and traffic features that are computed from the previous connections to the same host over a two-second time window.

The label column is called "label" and consist of the levels "normal." and 4 main types of attack: DOS, R2L, U2R and probing, which can be further divided into 24 subtypes. In our test we will not differentiate between them. We use the "normal." level for the normal data points and all the other classes as anomalies. For anomaly detection tests with this dataset we use a test set size of 10 000 samples and an anomaly ratio of 0.2.

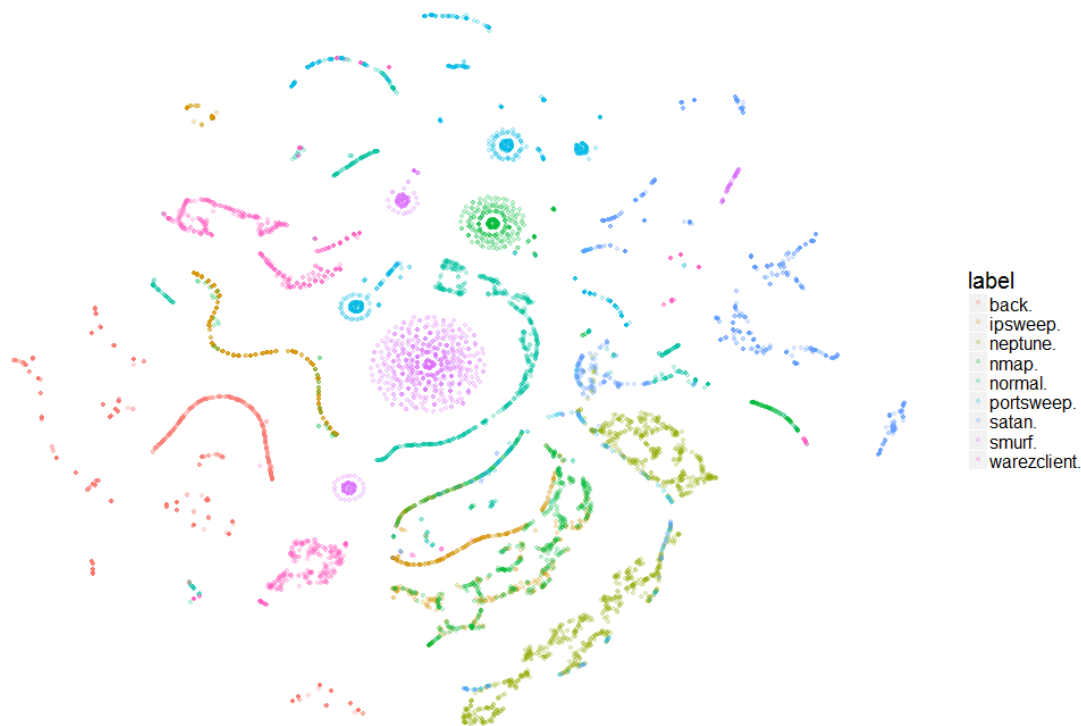


Figure 5.4: This figure shows a t-SNE embedding of the Network Intrusion dataset with $n = 9\,000$ samples (1 000 samples for each of the 9 most populated levels).

We can see that the normal points populate a very distinct region of the data space, barely mixed up with other points. Thus we expect the anomaly detection methods to achieve an AUC performance of almost 1.0, which is confirmed by the experiments later in this chapter.

³Description and dataset can be found at <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

Higgs Challenge

The Higgs Challenge dataset has been used in the Kaggle "Higgs Boson Machine Learning Challenge" and consists of 31 feature columns with 250 000 entries.⁴

The features contain quantities about the bunch collision as measured by the detector and quantities computed from the primitive features, which were selected by the physicists of the particle detector ATLAS.

The label column "Label" has values "b" and "s" which indicate background and signal respectively. We use the background samples as normal values and the signal as anomalies.

For anomaly detection tests with this dataset we use a test set size of 10 000 samples and an anomaly ratio of 0.1.



Figure 5.5: This figure shows a t-SNE embedding of the Higgs Challenge dataset with $n = 10\,000$ samples (5 000 samples for each class).

We can see in Figure 5.5, that the two classes are strongly mixed up. The only difference is, that certain regions are more populated by one class than by the other one. Thus we don't expect high AUC performances, which is confirmed by the experiments where the best methods perform around an AUC of 0.72.

⁴This dataset as well as the description are provided by the Kaggle homepage: <https://www.kaggle.com/c/higgs-boson/data>.

Comparison of the Datasets

The following table shows an overview of the dataset properties, such as the number of features p , the number of samples n , the number of categorical variables p_{cat} and the mean number of levels per categorical variable $\mu_{nlevels}$.⁵

Table 5.1: A comparison of the dataset properties.

Dataset	p	n	p_{cat}	$\mu_{nlevels}$
Forest Cover Type	52	581 012	44	2
Credit Card Fraud	29	284 807	0	-
Financial Data	9	1 056 320	5	8.4
Network Intrusion	41	4 898 431	9	10.7
Higgs Challenge	31	250 000	0	-

5.2 Testing Pipeline

The testing pipeline we use for the tests follow the schema shown in Figure 5.6. The first step the test program is to load the dataset and to split it randomly into a train and a test set, according to some predefined parameters, e.g sizes of training and test set, anomaly ratio, name of the label column or names of the normal/anomaly levels in the label column.

The next step consists of fitting a representation model, using the test data. This option is not always used, but to keep the test program simple we handle this case with an "Identity representation model" which just returns the input unchanged. After that, the train and the test set get transformed into new train and test sets.

The actual anomaly detection happens in the third step. We fit the anomaly detection model, using the (transformed) train data. After the fitting process, the program computes the anomaly scores for the test data and compares them with the real labels using the AUC measure. Note that the program knows the actual labels of the test set, since it created the test set according to the known levels of the label column.

The implementation of the anomaly and representation methods and the whole pipeline itself was a complex endeavour and needs some explanation to understand it in detail. Thus, for implementation details of the methods, the pipeline as well as the control options of the testing function, we refer to the Appendix A.

⁵Description and dataset can be found at <https://www.kaggle.com/c/higgs-boson>.

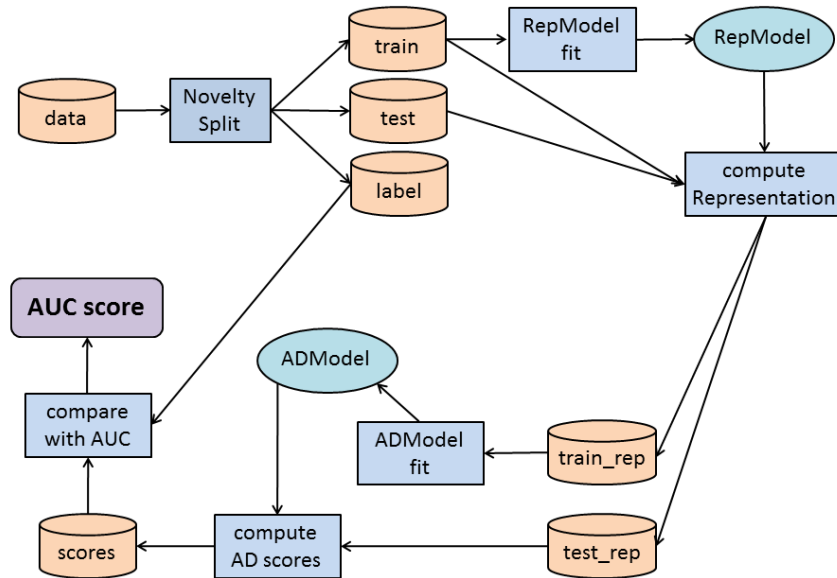


Figure 5.6: This picture shows the schema describing the different steps, needed to perform an anomaly detection test using a representation method. Note that the schema also allows to skip the representation part by replacing the representation model by a dummy model that simply passes on the data as it is.

5.3 Parallelization

Actual scenarios often consist of a lot of training samples, thus it is essential for the anomaly detection and representation methods to be parallelizable, in order to use all computational resources available.

With exception of the OSVM method, all methods are potentially parallelizable or already implemented to use multiple cores. Since the methods of the *H2O.ai* stack support by default multiple cores, the methods AE, KMD and KMC run already in parallel.

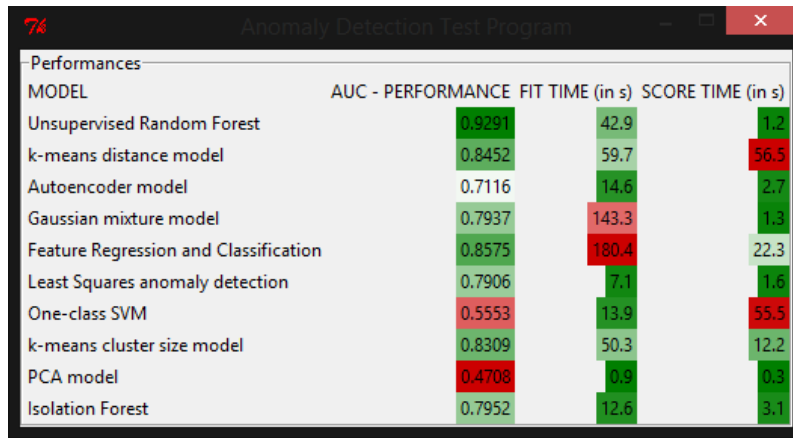
Furthermore the methods IF and URF mainly need to build independent building trees. Thus it is an easy task to parallelize this two methods. The *sklearn*-implementation we use for this two methods, already runs with multiple cores.

For the implementation of FRaC we used random forests which are themselves parallelizable. Further we have to build many models at once, which makes FRaC a perfect target for more parallelization, by building the different feature-models in parallel. PPCA consists of calculating an SVD decomposition and matrix multiplications which can be parallelized if necessary.⁶ The same is true for the LSAD method which consists of matrix multiplications and a matrix inversion, which is done using an LU decomposition. Both processes can be parallelized. For the GMM there exist parallel implementations as well (Kwedlo, 2014).

Using parallelization, the methods we tested all perform satisfyingly fast. An example of the fit and scoring times of a test on the Forest Cover Type dataset with original features

⁶A parallel implementation of PCA is provided by *H2O.ai*: <http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/pca.html>

and $n = 100\,000$ training and $n = 50\,000$ test samples, can be seen in Figure 5.7.



The screenshot shows a window titled 'Anomaly Detection Test Program' with a 'Performances' tab. It displays a table of results for ten different models. The columns are 'MODEL', 'AUC - PERFORMANCE', 'FIT TIME (in s)', and 'SCORE TIME (in s)'. Values are color-coded: green for good performance/low time and red for poor performance/high time.

MODEL	AUC - PERFORMANCE	FIT TIME (in s)	SCORE TIME (in s)
Unsupervised Random Forest	0.9291	42.9	1.2
k-means distance model	0.8452	59.7	56.5
Autoencoder model	0.7116	14.6	2.7
Gaussian mixture model	0.7937	143.3	1.3
Feature Regression and Classification	0.8575	180.4	22.3
Least Squares anomaly detection	0.7906	7.1	1.6
One-class SVM	0.5553	13.9	55.5
k-means cluster size model	0.8309	50.3	12.2
PCA model	0.4708	0.9	0.3
Isolation Forest	0.7952	12.6	3.1

Figure 5.7: This figure shows a the results of an anomaly detection test with all covered methods, using $n = 100\,000$ training samples and $n = 50\,000$ test samples of the Forest Cover Type dataset.

We can recognize that the fitting times range from 0.9 to 180.4 seconds and the scoring times from 0.3 to 56.5 seconds. These are reasonable fast times for an actual working novelty detection pipeline in service. A fitting time of 180 seconds doesn't present a problem and the maximal scoring time per sample of 0.001 seconds isn't an issue either.

5.4 Learning Curves with Original Data

In this test we want to find out, how the different anomaly detection algorithms perform compared with each other, using varying numbers of training samples without any representation.

Learning Curves - Forest Cover Type

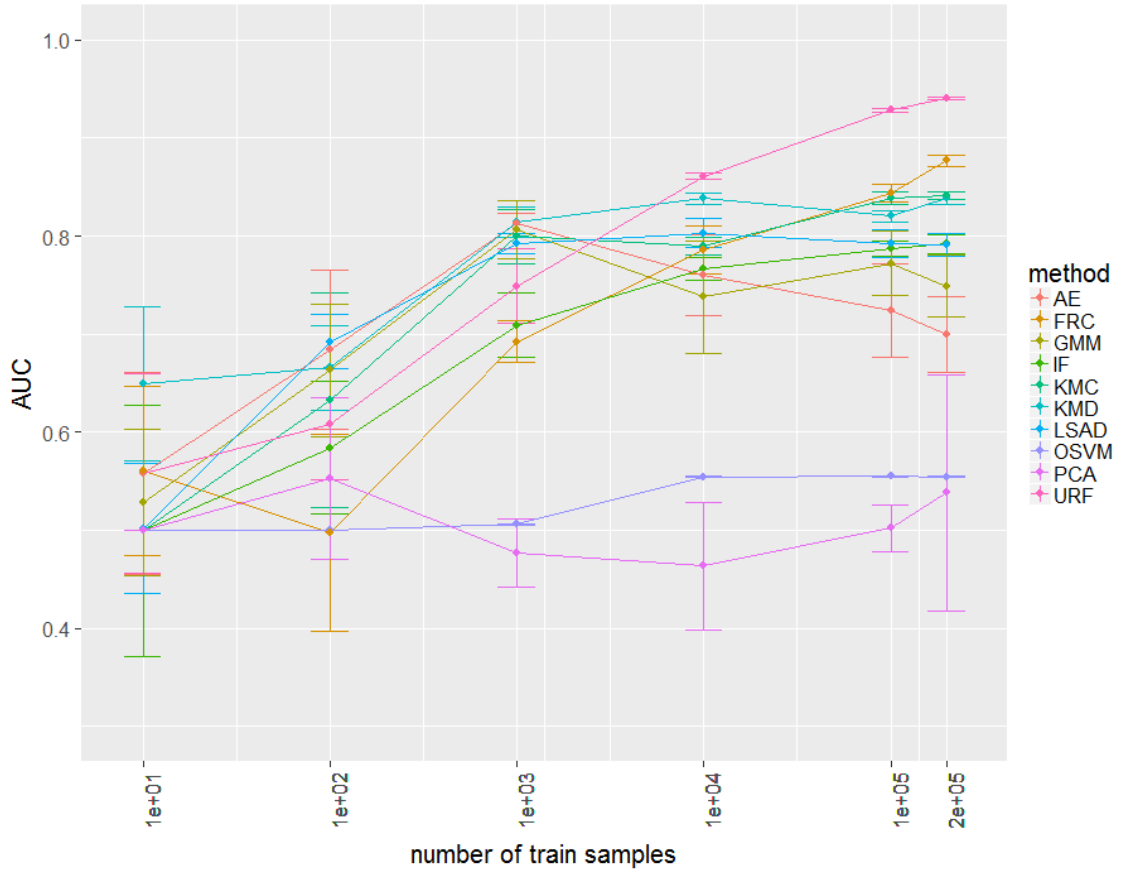


Figure 5.8: Performance of the different anomaly detection methods on the Forest Cover Type dataset, using $n = 10, 100, 1\,000, 10\,000, 100\,000, 200\,000$ training samples.

The test shows that for a small number of samples the KMC and LSAD methods outperform the other methods whereas for a higher number of samples ($\geq 10\,000$) the URF method has a clear edge. Overall one can say that all algorithms, with exception of OSVM and PCA, are able to perform considerably better than random ($AUC \geq 0.5$) for $\geq 1\,000$ samples.

Learning Curves - Credit Card Fraud

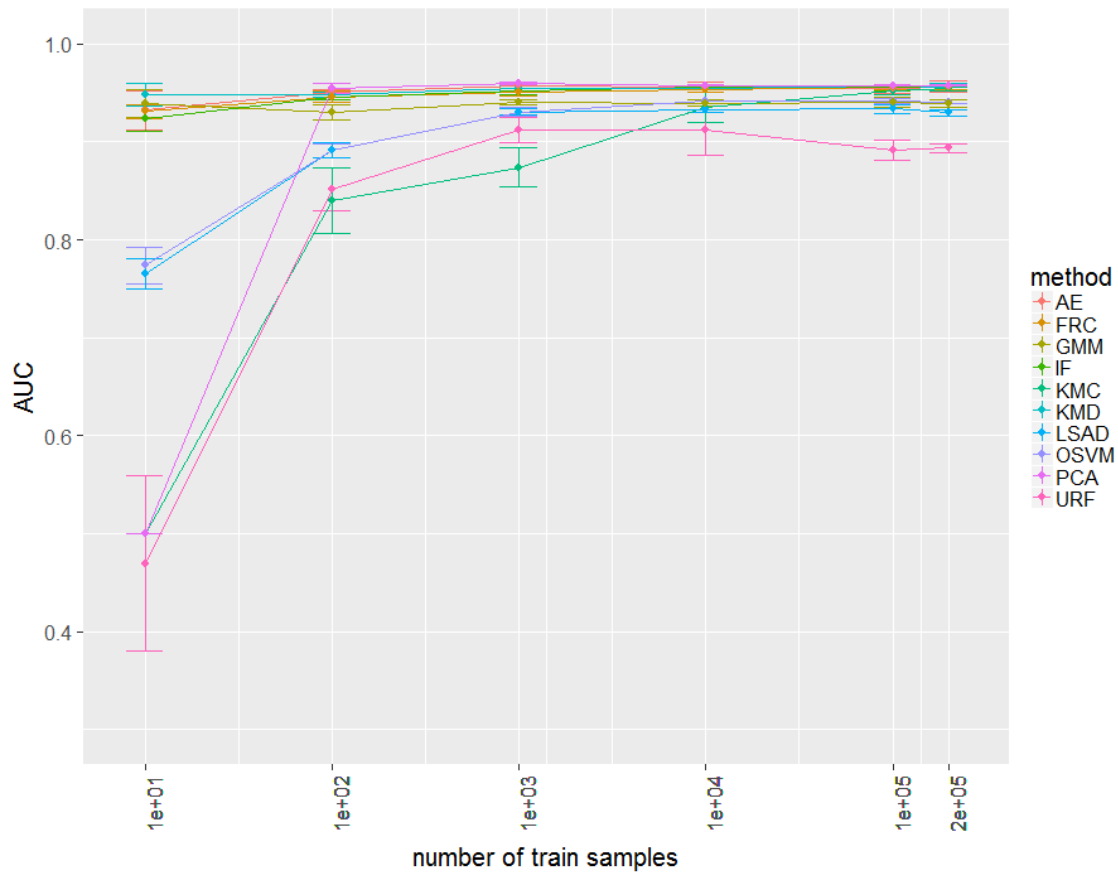


Figure 5.9: Performance of the different anomaly detection methods on the Credit Card Fraud dataset, using $n = 10, 100, 1\,000, 10\,000, 100\,000, 200\,000$ training samples.

We can see that all methods, with exception of URF and KMC, are able to achieve a high performance. The differences are generally very small for the other algorithms. In addition we see that most methods don't learn any more after 1 000 samples.

Learning Curves Curves - Financial Data

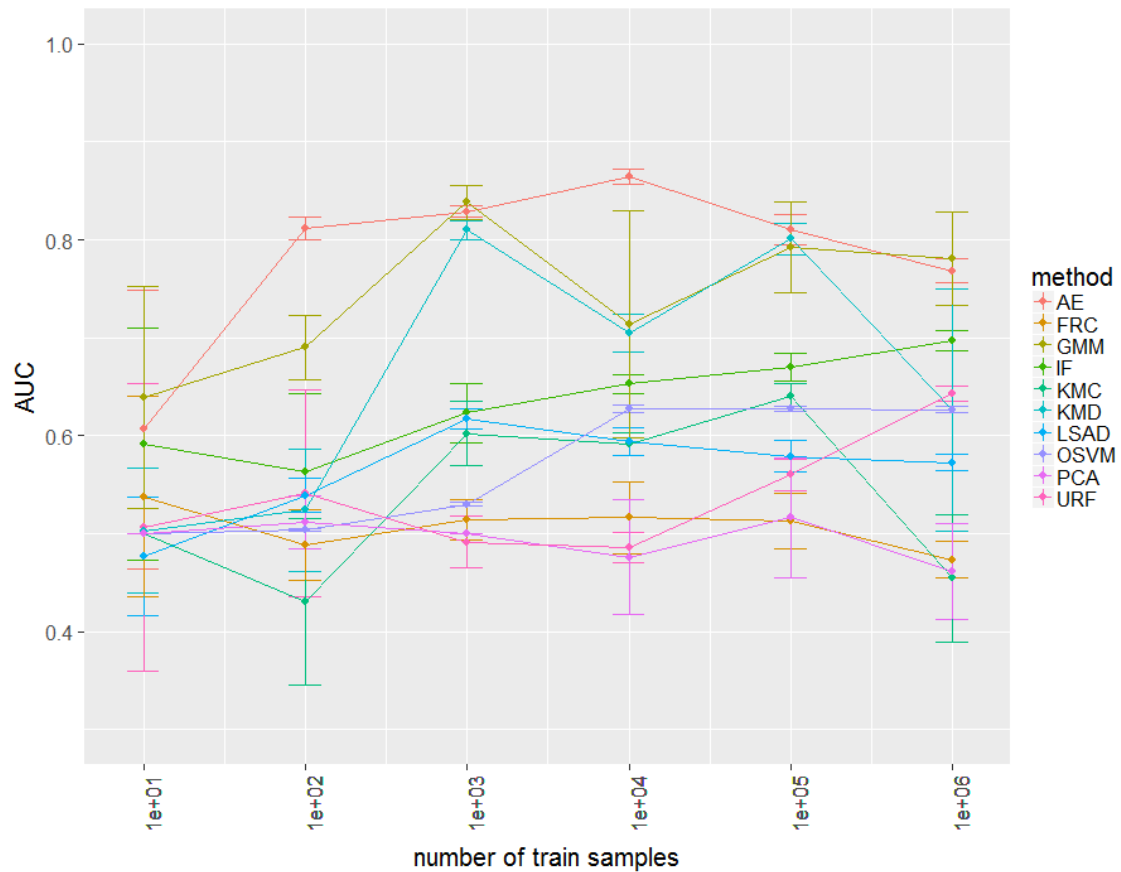


Figure 5.10: Performance of the different anomaly detection methods on the Financial Dataset, using $n = 10, 100, 1\,000, 10\,000, 100\,000, 1\,000\,000$ training samples.

The Financial Dataset shows strongly varying performances between the methods. The best performing methods are AE, GMM and KMC. On the other hand PCA, URF and FRC perform not or barely better than random. All methods with exception of KMC don't learn anymore after $\geq 10\,000$ samples.

Learning Curves - Network Intrusion

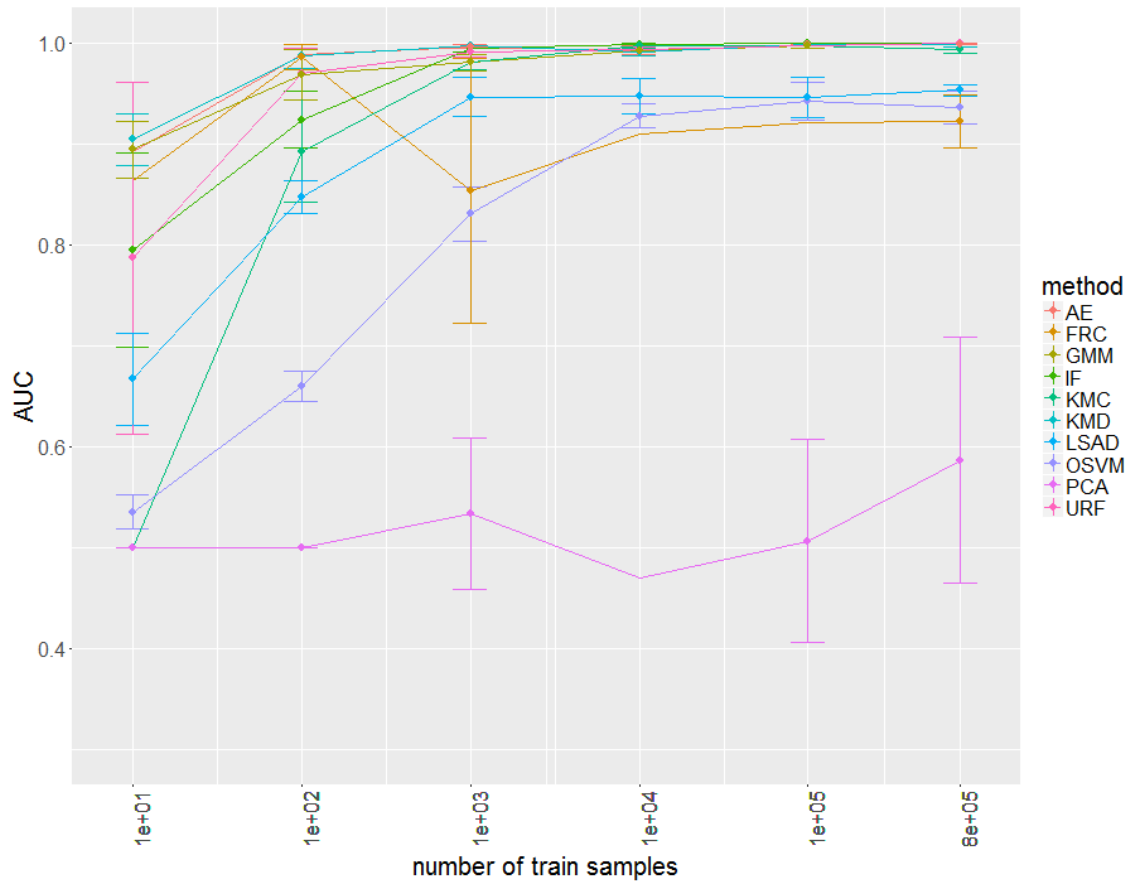


Figure 5.11: Performance of the different anomaly detection methods on the Network Intrusion dataset, using $n = 10, 100, 1\,000, 10\,000, 100\,000, 1\,000\,000$ training samples.

The performance of the different algorithms strongly varies among the methods for the Intrusion Detection dataset. The method PCA doesn't learn anything and methods like OSVM and LSAD need a moderate amount of samples ($\geq 1\,000$), to have a performance comparable to the other methods. In contrast, methods like IF, URF, AE or GMM achieve an average AUC performance of almost 1 (> 0.98) for only 1 000 samples.

Learning Curves - Higgs Challenge

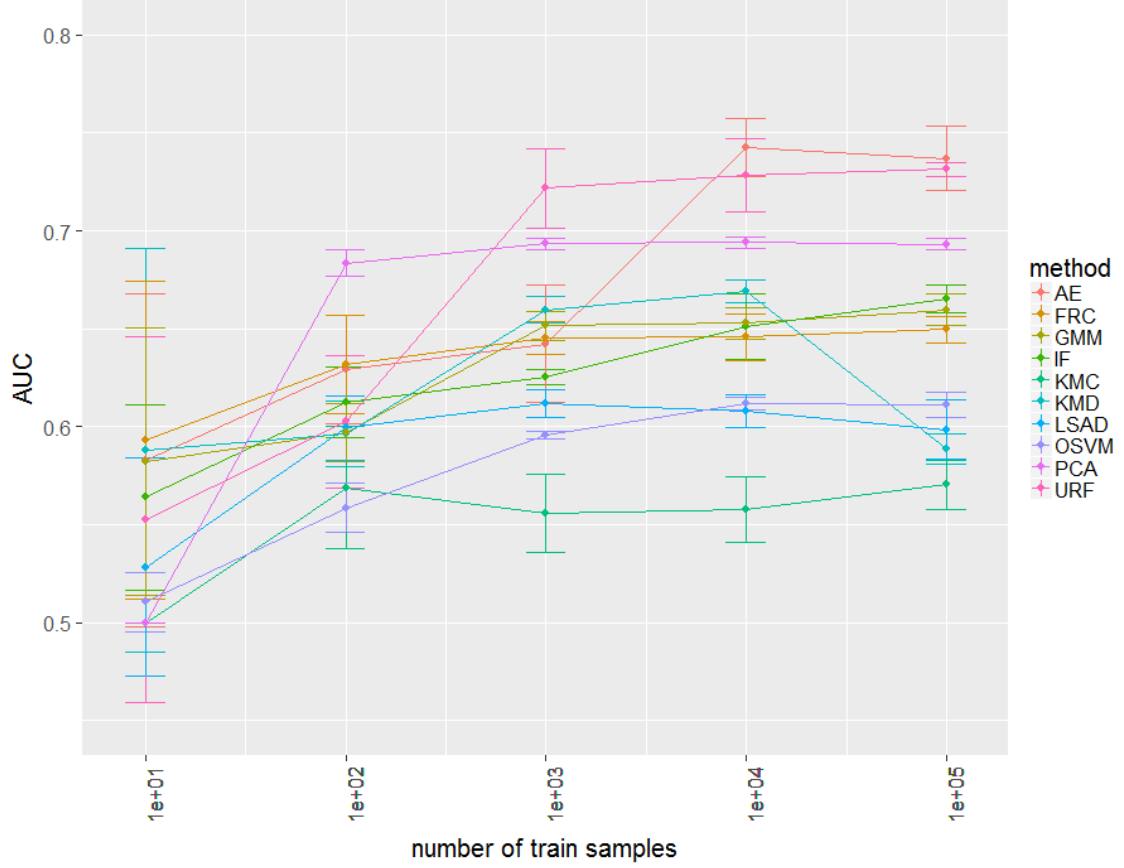


Figure 5.12: Performance of the different anomaly detection methods on the Higgs Challenge dataset, using $n = 10, 100, 1\,000, 10\,000, 100\,000$ training samples.

The different methods show strongly varying performances, but all perform better than random for ≥ 100 samples. For 100 samples PCA has quite an advantage over the other methods but gets outperformed by URF for 1 000 samples, which itself gets out-shined by the AE method for $\geq 10\,000$ samples. Only small performances are achieved by KMC, OSVM and LSAD.

One can see that the behaviour of the methods on the different datasets deviate strongly from each other. Nevertheless certain things can be said that seem to hold over almost all datasets. For example at the small end of the number of training samples, there seem to be some clear winners. The methods KMD, GMM, AE frequently outperform the other methods. In contrast there are methods that give moderate performance over the complete range for all datasets, such as GMM, IF or KMD.

5.5 Compression Curves with PCA Represented Data

This and the next two sections 5.6 and 5.7 illustrate how the anomaly detection models work together with the different representations. The key factor that influences this interaction, is the amount of compression that happens in the representation step. We formalize this idea by introducing the compression factor, which is defined as

$$factor := \frac{\text{number of representation columns}}{\text{number of original columns (dummy encoded)}}. \quad (5.5.0.1)$$

Note that we use as reference number, the number of columns when the original dataset gets dummy encoded. The reason for this is that all the representation algorithms handle the categorical variables with dummy encoding and thus interpreting the value of the denominator in (5.5.0.1) as the size of a "full" representation, makes sense.

We want to find out, which choices of compression factors are sensible for different "representation + detection method"-combinations. Thus we test how the different anomaly detection algorithms perform, when we vary the parameter factor for the representation method.

In principle, certain representation methods, such as the Autoencoder and Entity Embedding allow for a representation factor ≥ 1.0 . But we will stick to the case $(0, 1]$ in our experiments, such that we can compare the results to other representation methods which are bounded to a compression factor smaller or equal to one, such as PCA.

PCA Compression Curves - Forest Cover Type

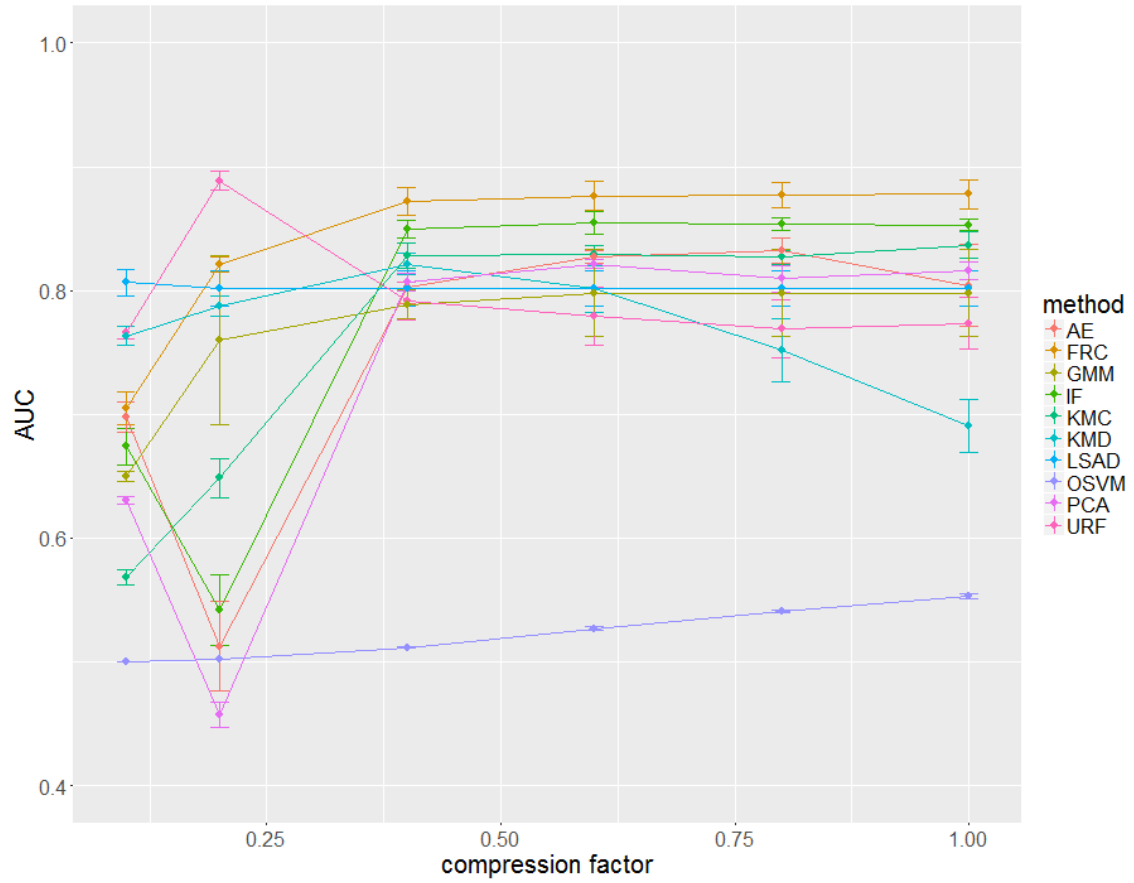


Figure 5.13: Performance of the different anomaly detection methods on the Forest Cover Type dataset with PCA represented data, using $n=10\,000$ training samples and compression factors ranging from 0.1 to 1.0.

When we look at the performances of the methods with respect to a varying compression factor we recognize, that the different methods show different behaviours. The methods FRaC, IF, KMC, AE, PCA, LSAD and GMM are almost constant in performance on different factors. Other methods like KMD, URF and OSVM show a clear curve, having a maximal performance at compression factor values 0.2, 0.4 and 1.0 respectively.

PCA Compression Curves - Credit Card Fraud

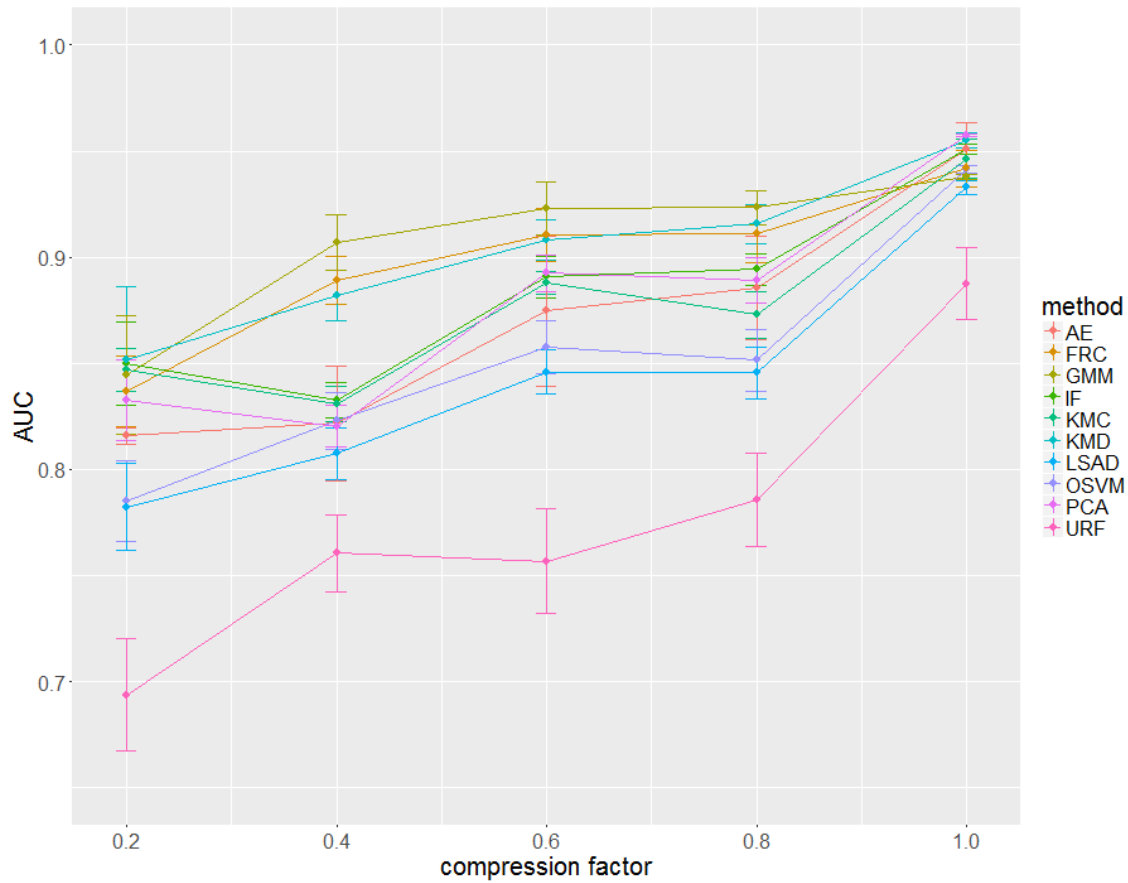


Figure 5.14: Performance of the different anomaly detection methods on the Credit Card Fraud data with PCA represented data, using $n=10\,000$ training samples and compression factors ranging from 0.2 to 1.0.

The methods show an almost monotonously increasing curve, where all curves have approximately the same behaviour. This result is not unexpected, if we remember from the description of the Credit Card Fraud dataset, that the data itself consisted mainly of PCA components itself. Thus we can expect that useless dimensions have already been cut away and by using the complete PCA encoded representation, the algorithms have the best detection performances.

PCA Compression Curves - Financial Data

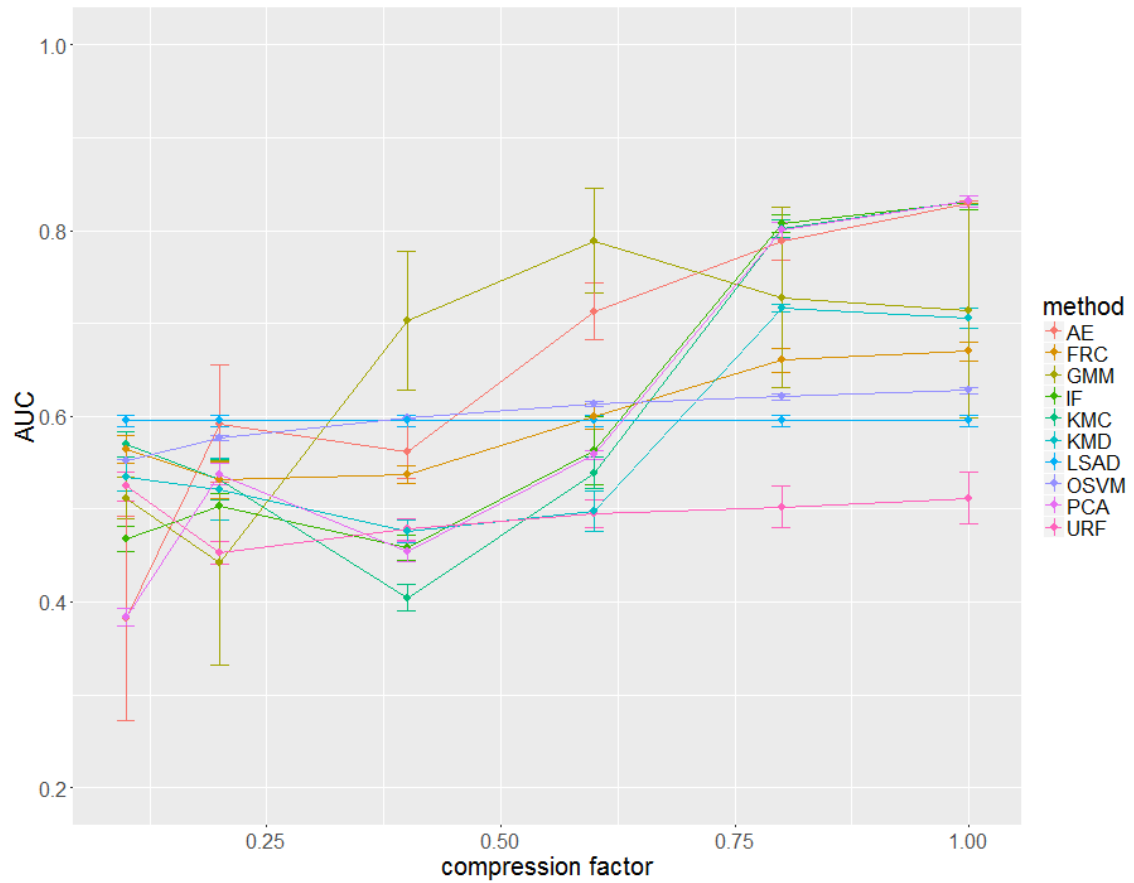


Figure 5.15: Performance of the different anomaly detection methods on the Financial Dataset with PCA represented data, using $n=10\,000$ training samples and compression factors ranging from 0.1 to 1.0.

The performances of the methods OSVM and LSAD are almost constant with varying compression factor and GMM has peak performance at 0.4. All the other algorithms have their peak performance at factor 1.0.

PCA Compression Curves - Network Intrusion

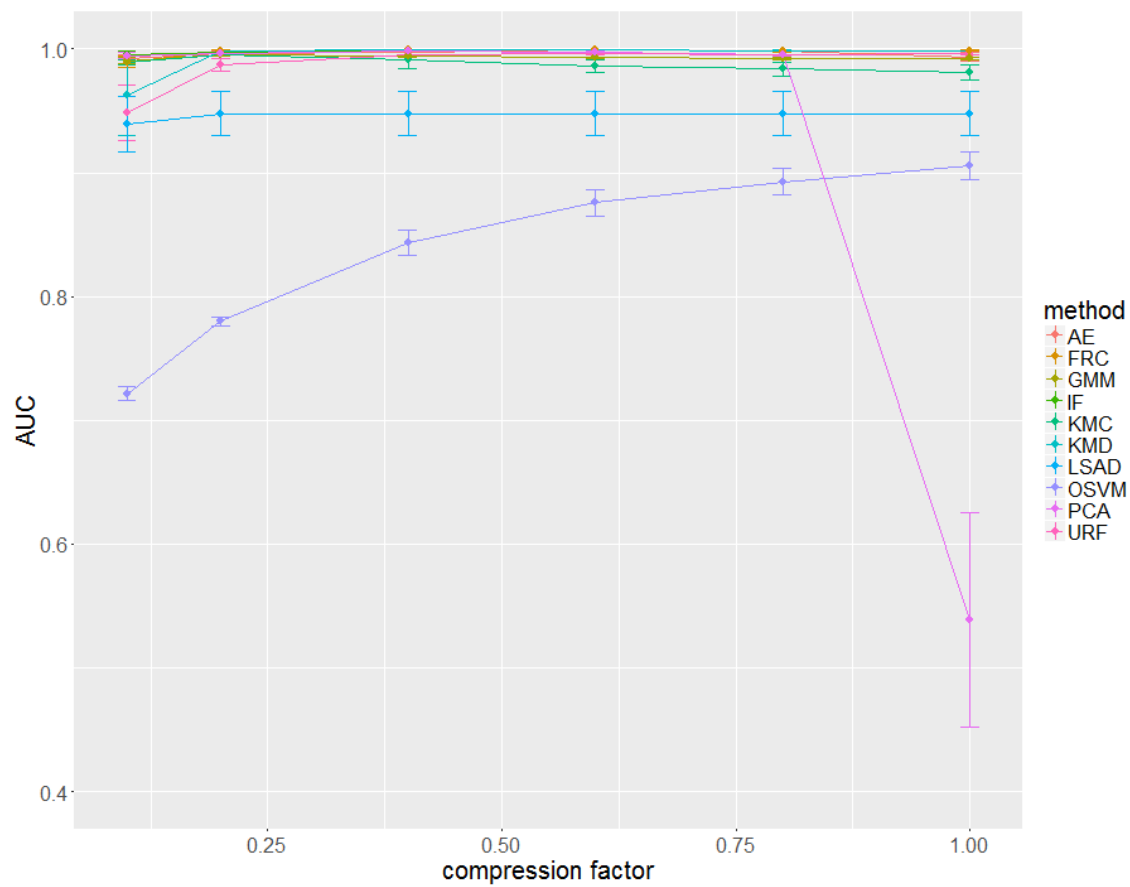


Figure 5.16: Performance of the different anomaly detection methods on the Financial Dataset with PCA represented data, using $n=10\,000$ training samples and compression factors ranging from 0.1 to 1.0.

All the methods perform approximately equally over the complete range of compression factors, with exception of OSVM, which increases performance with increasing factor and PCA, which performs drastically bad for a compression factor of 1.0

PCA Compression Curves - Higgs Challenge

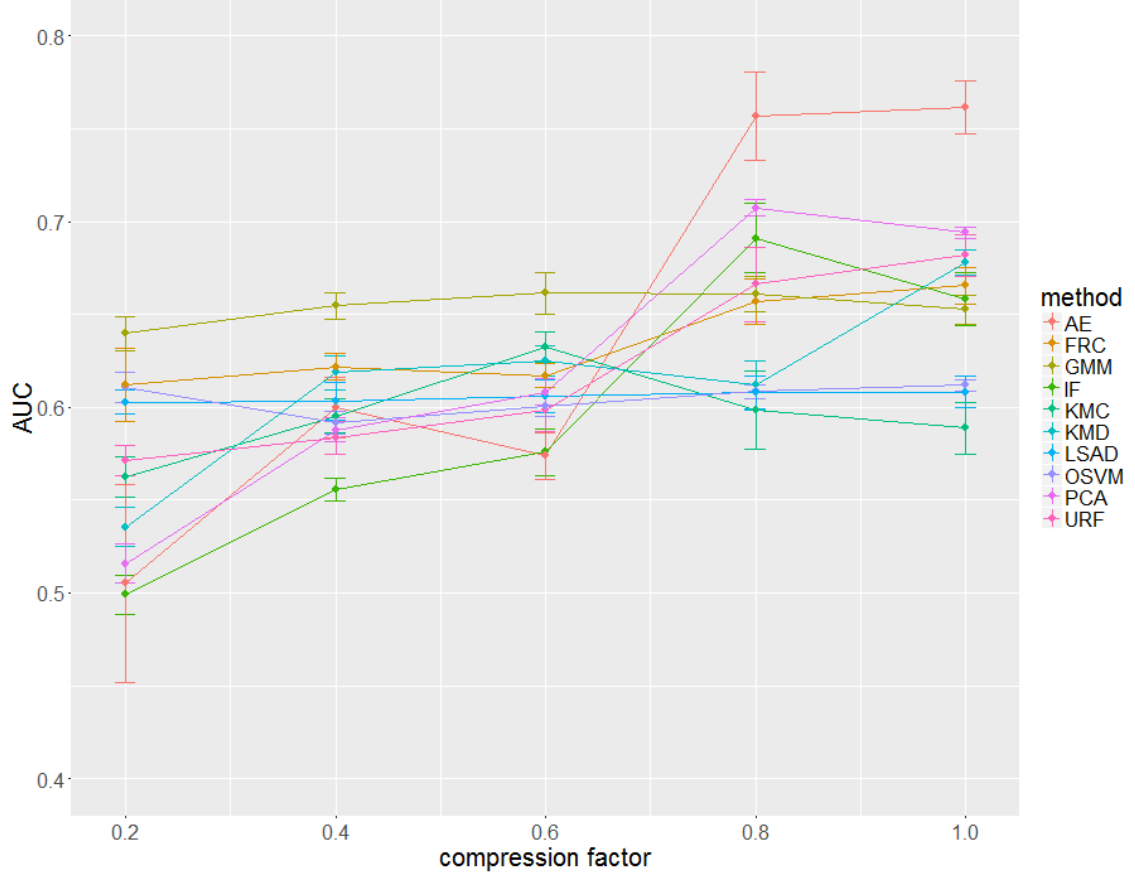


Figure 5.17: Performance of the different anomaly detection methods on the Higgs Challenge dataset with PCA represented data, using $n=10\,000$ training samples and compression factors ranging from 0.2 to 1.0.

For the Higgs Challenge set we recognize that KMC and GMM show a weak curve with peak at 0.6. Further the methods AE, PCA, URF, FRC and IF show a considerable jump in performance at 0.8. The methods OSVM and LSAD are almost constant with varying compression factor.

To conclude we can say that different methods have different optimal compression factors. However the optimal compression factor is not always the same for a method over all datasets. The methods GMM and KMD seem to perform good with compression factors of 0.6. Other methods generally increase performance with increasing compression factor and thus have maximal performance at factors 0.8 – 1.0, such as AE, FRC, IF, KMC, OSVM, PCA and URF. A special case builds the LSAD method, because it shows for all datasets an almost constant performance with respect to a varying compression factor.

5.6 Compression Curves with AE Represented Data

AE Compression Curves - Forest Cover Type

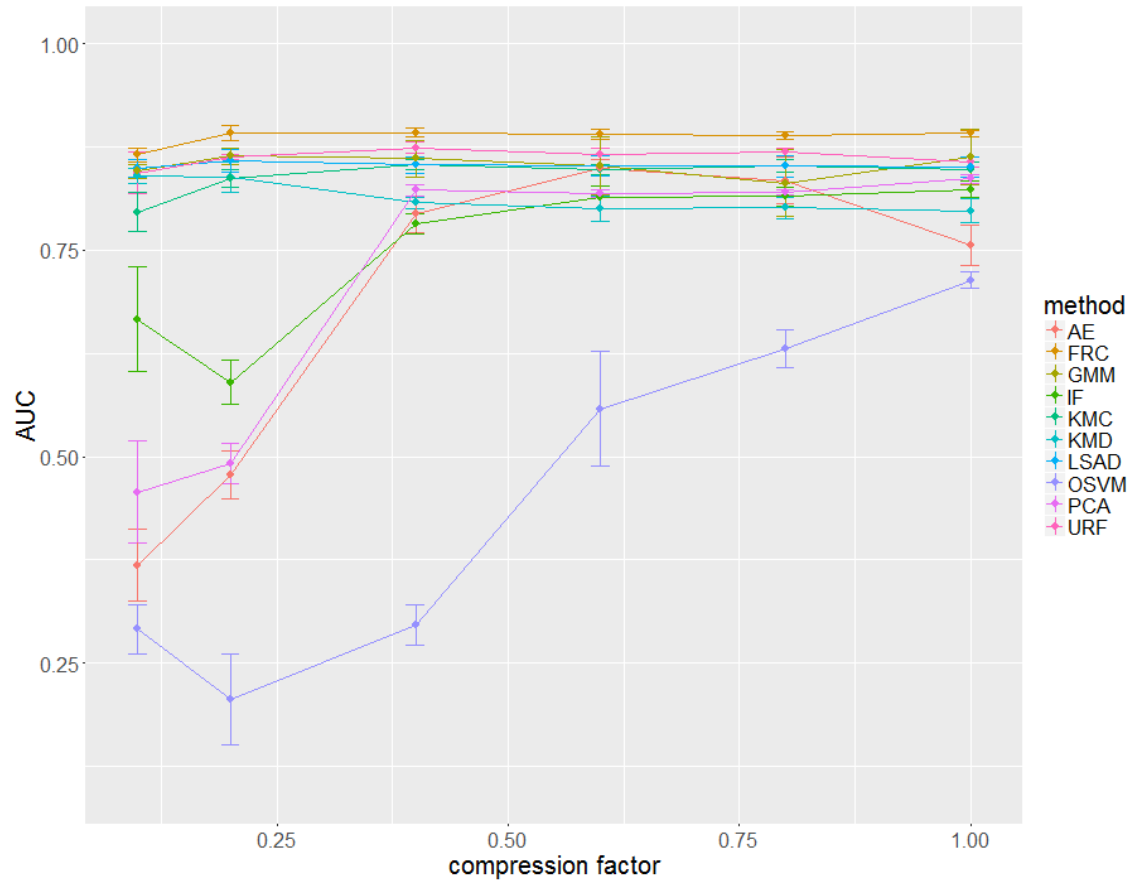


Figure 5.18: Performance of the different anomaly detection methods on the Forest Cover Type dataset with AE represented data, using $n=10\,000$ training samples and compression factors ranging from 0.1 to 1.0.

The different methods interact differently with the varying compression factor. Methods like OSVM, AE, IF and PCA show a curved behaviour with respect to the compression factor, the other methods are almost stable.

AE Compression Curves - Credit Card Fraud

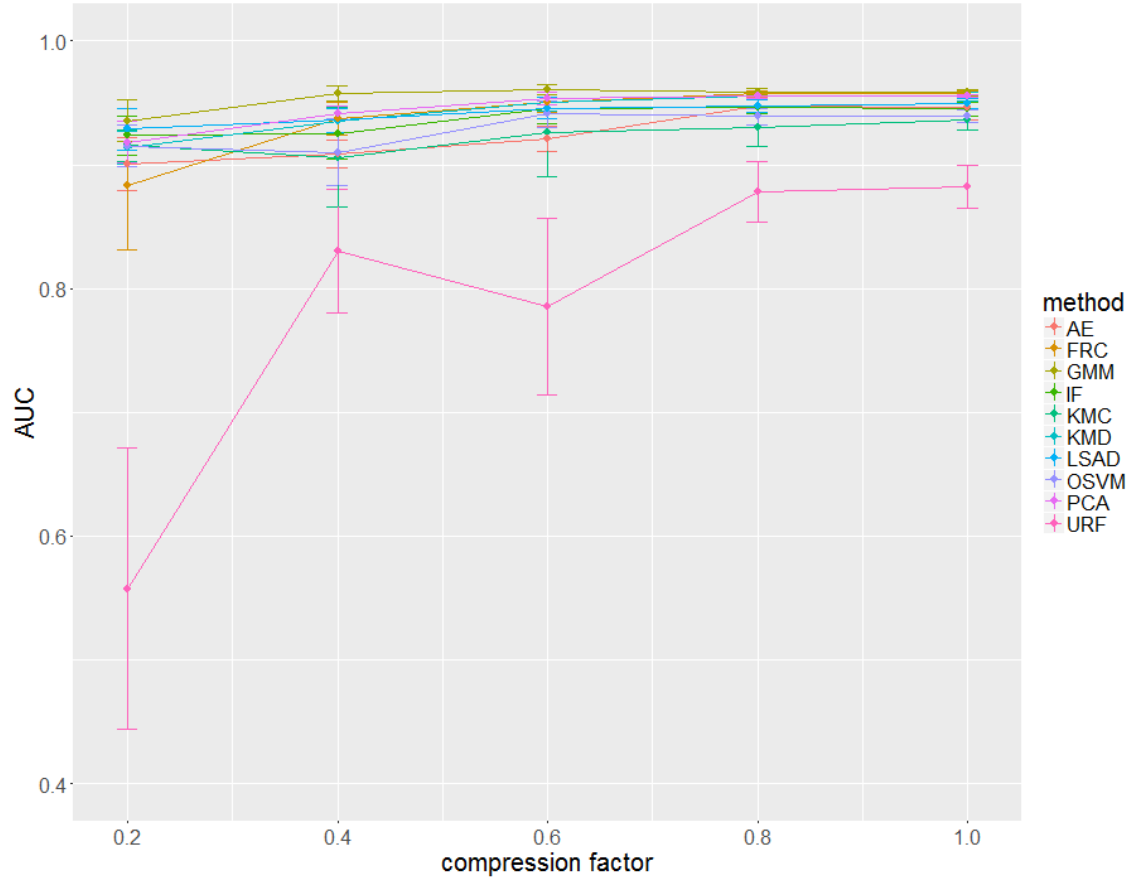


Figure 5.19: Performance of the different anomaly detection methods on the Credit Card Fraud dataset with AE represented data, using $n=10\,000$ training samples and compression factors ranging from 0.2 to 1.0.

All the methods are either stable or increase their performance with increasing compression factor. This behaviour is not new as we have already seen in Figure 5.14. the compression curve of the same dataset with PCA represented data, showed that more information generally means an improvement in performance for all methods.

AE Compression Curves - Financial Data

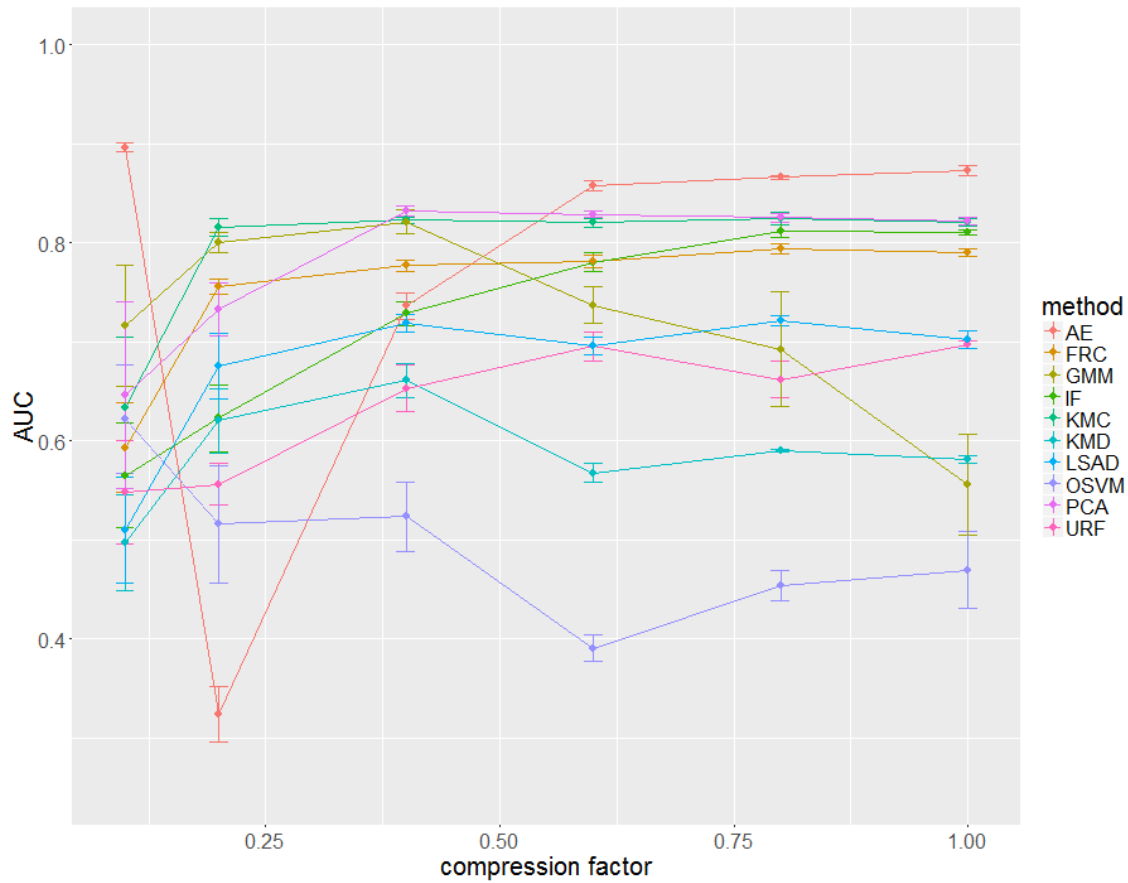


Figure 5.20: Performance of the different anomaly detection methods on the Financial Dataset with AE represented data, using $n=10\,000$ training samples and compression factors ranging from 0.1 to 1.0.

For this dataset, we can see again two types of behaviour. Methods like AE, FRC, IF, KMC, LSAD, PCA and URF stay constant or increase performance with bigger compression factors whereas methods like GMM, KMD and OSVM show a clear peak performance at 0.4, 0.4 and 0.1 respectively. The huge performance of the AE with factor 0.1 is probably due to some special effects that we expect won't generalize and thus we ignore.

AE Compression Curves - Network Intrusion

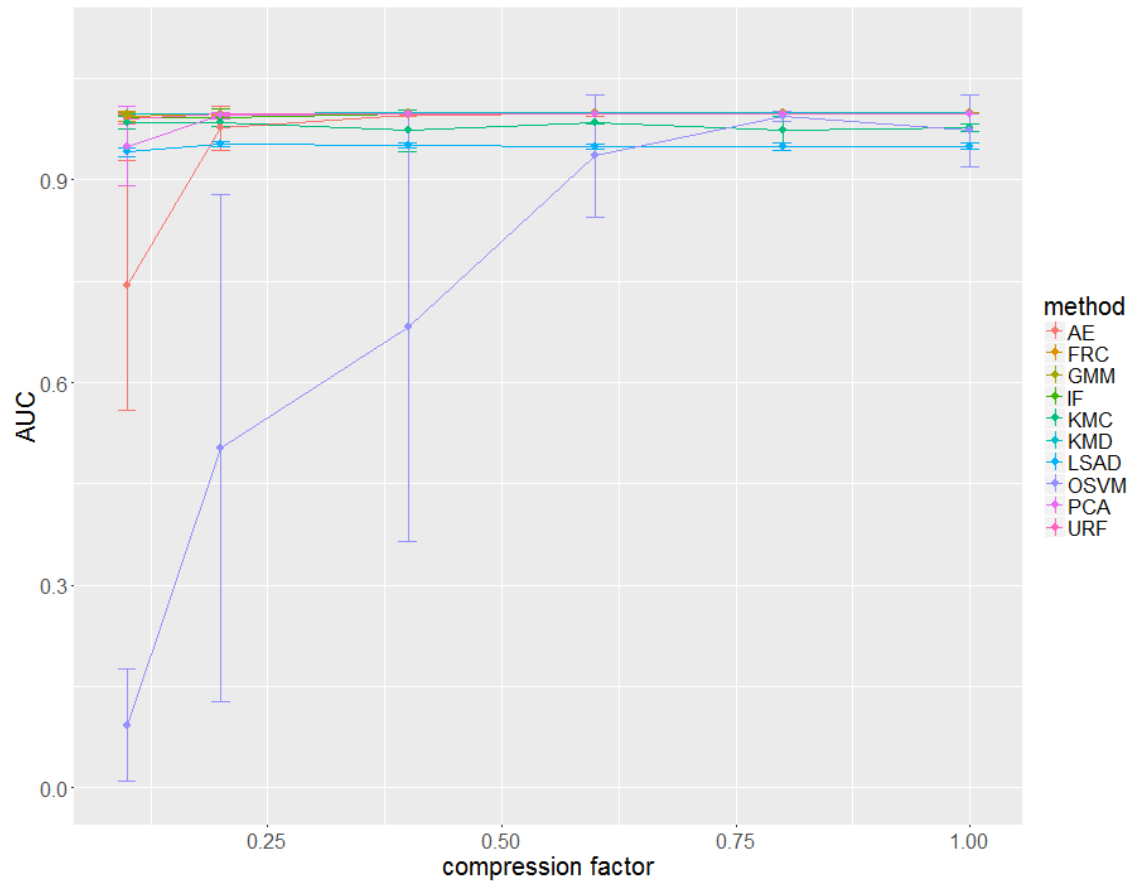


Figure 5.21: Performance of the different anomaly detection methods on the Network Intrusion Dataset with AE represented data, using $n=10\,000$ training samples and compression factors ranging from 0.1 to 1.0.

For this dataset almost all methods with exception of OSVM and AE, perform approximately constant with varying compression factor.

AE Compression Curves - Higgs Challenge

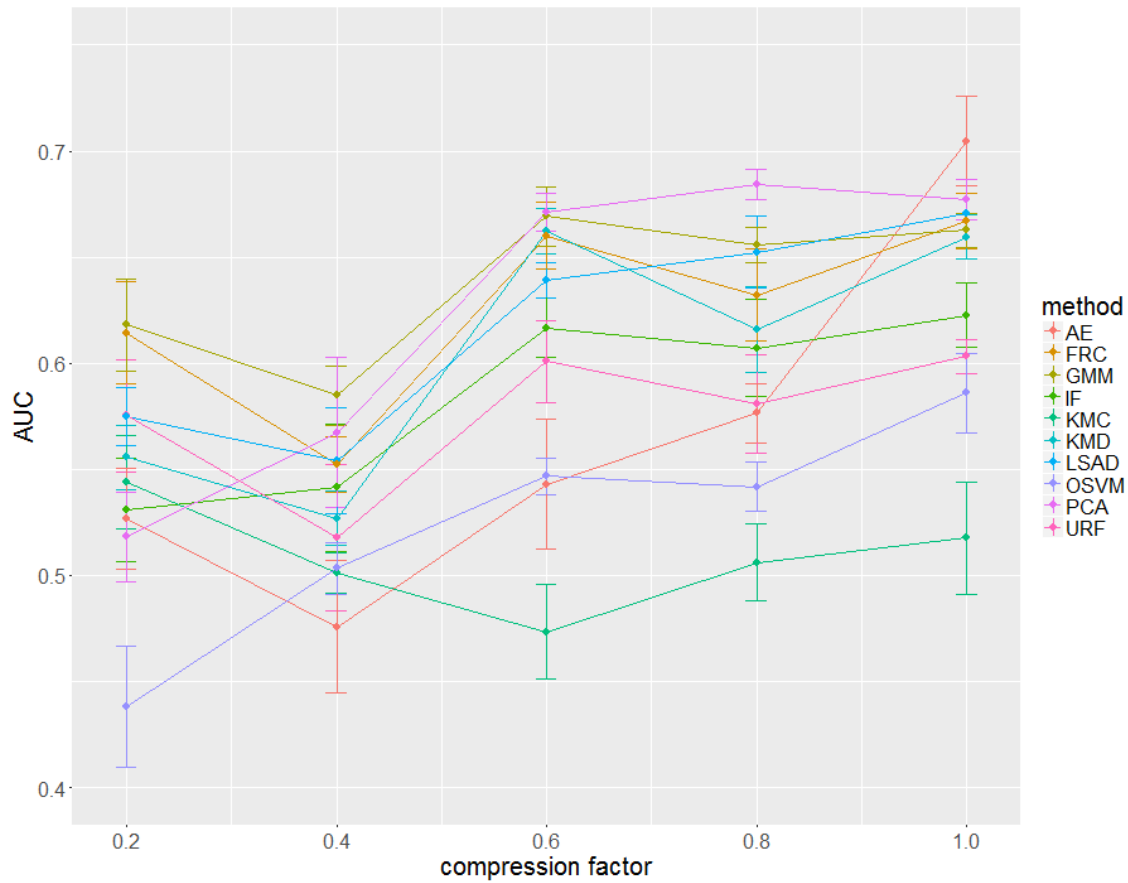


Figure 5.22: Performance of the different anomaly detection methods on the Higgs Challenge Dataset with AE represented data, using $n=10\,000$ training samples and compression factors ranging from 0.2 to 1.0.

Almost all methods show an approximately monotonous increase in performance with increasing compression factor. The decreasing tendencies of all methods at 0.4 and 0.8 we expect are due to some effect from the AE representations which are slightly worse in this cases by chance and thus don't reflect a real effect in the methods behaviour.

To conclude we can say, that the methods behave differently with varying compression factor. The methods FRaC, IF and URF perform best with a compression factor of 1.0, the methods GMM and KMC perform best for factors 0.2-0.4. The PCA method shows the best performance at 0.8. As already the case for the PCA representation, the LSAD method shows almost constant performance over all compression factors ≥ 0.2 . The two methods AE and OSVM show different behaviours over the different datasets.

5.7 Compression Curves with EE Represented Data

EE Compression Curves - Forest Cover Type

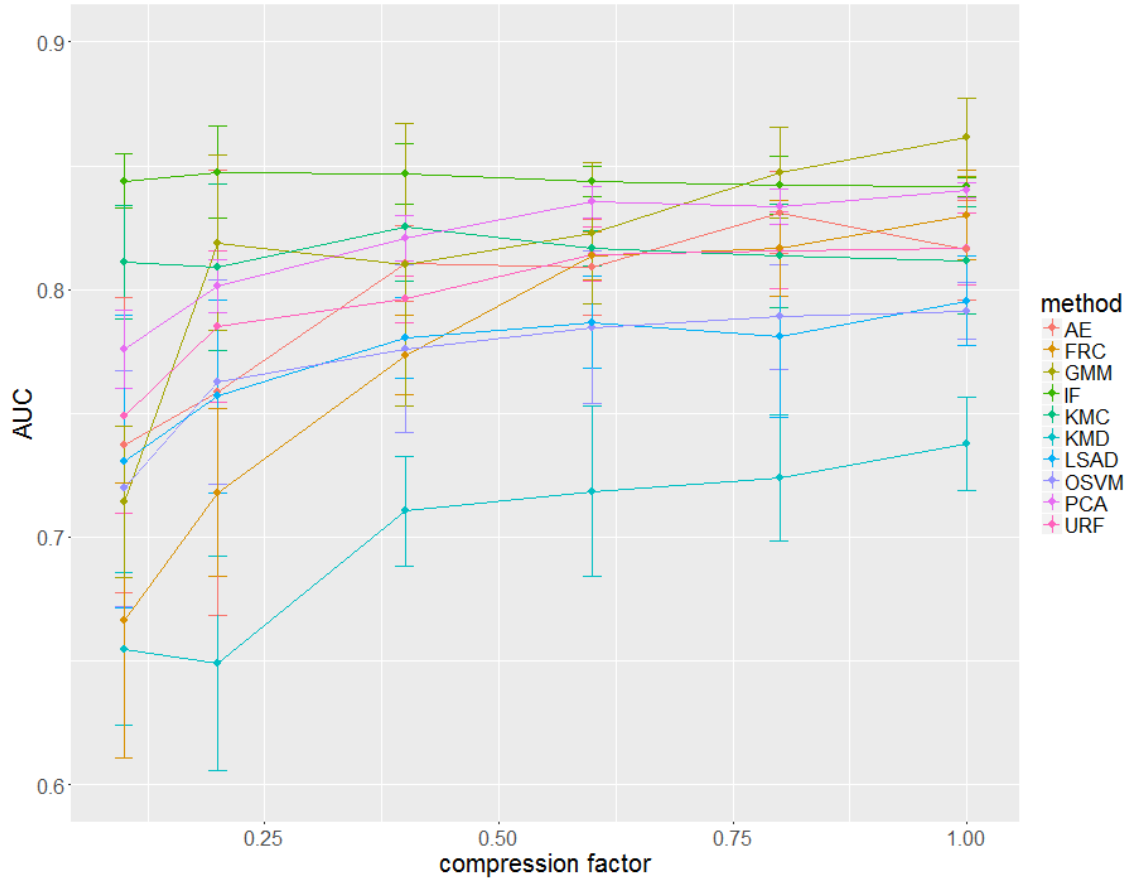


Figure 5.23: Performance of the different anomaly detection methods on the Forest Cover Type dataset with EE represented data, using $n=10\,000$ training samples and compression factors ranging from 0.1 to 1.0.

We can observe different behaviours of the methods with respect to a varying compression factor of the EE representation method. A number of methods, FRC, GMM, KMD, LSAD, OSVM, PCA and URF show an increasing performance with increasing compression factor. Further there are the methods AE, IF which show a clear peak at 0.25 and KMC that performs constant over the whole range.

EE Compression Curves - Credit Card Fraud

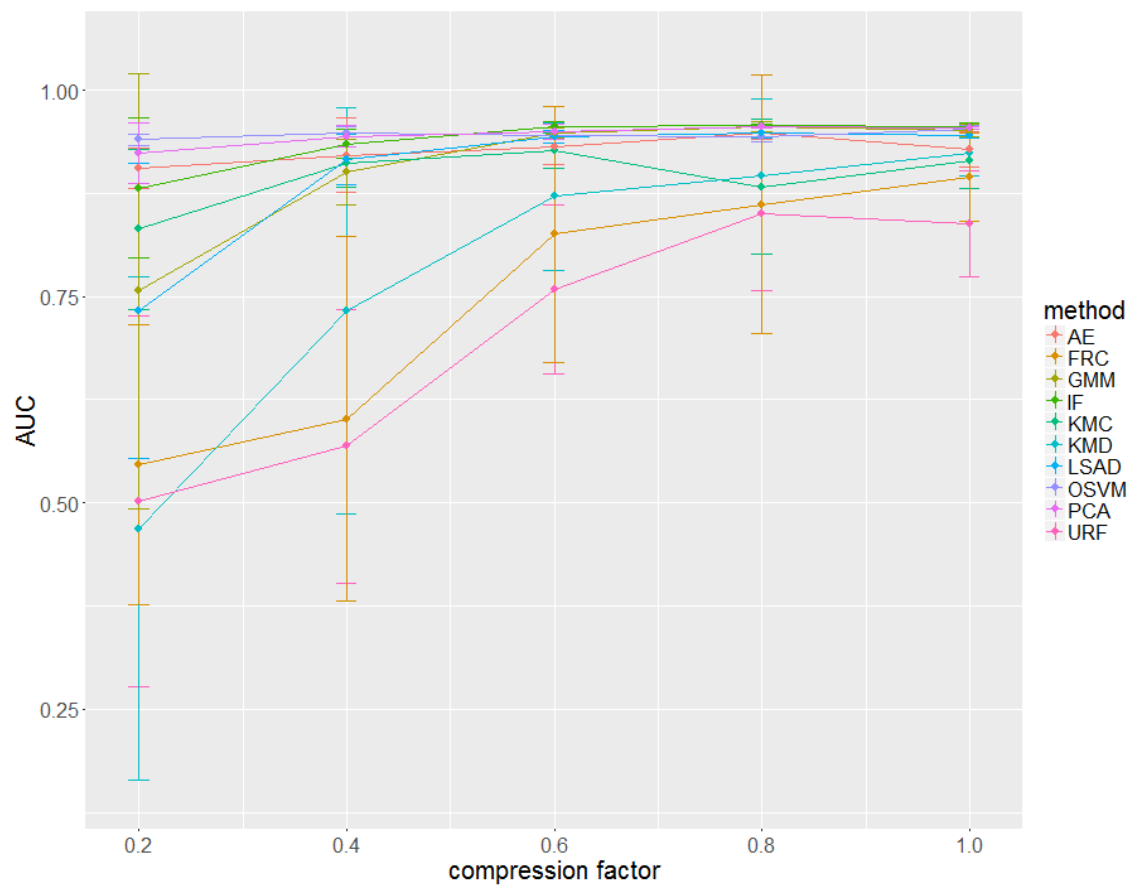


Figure 5.24: Performance of the different anomaly detection methods on the Credit Card Fraud dataset with EE represented data, using $n=10\,000$ training samples and compression factors ranging from 0.2 to 1.0.

For this dataset, almost all methods show an increasing performance in the compression factor. The methods KMC and AE build an exception, because they show a clear peak at 0.6 and 0.8 respectively.

EE Compression Curves - Financial Data

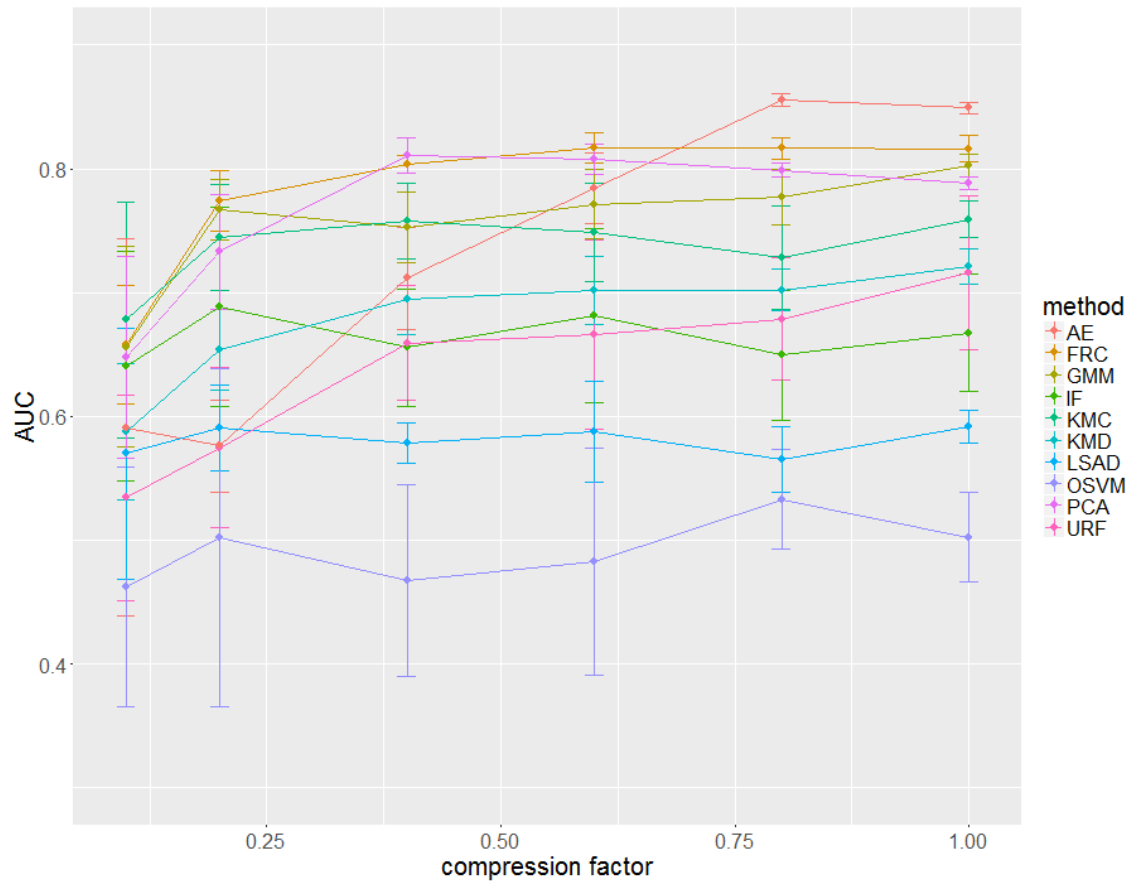


Figure 5.25: Performance of the different anomaly detection methods on the Financial Dataset with EE represented data, using $n=10\,000$ training samples and compression factors ranging from 0.1 to 1.0.

This dataset shows overall two types of behaviours, methods that show constant performance and methods, that show increasing performance with respect to increasing compression factor. To the first type belong the methods AE, FRC, GMM, KMD, URF and to the second one the methods IF, KMC, LSAD, OSVM. The PCA method an exception to this two types, because it shows a clear peak at 0.4.

EE Compression Curves - Network Intrusion

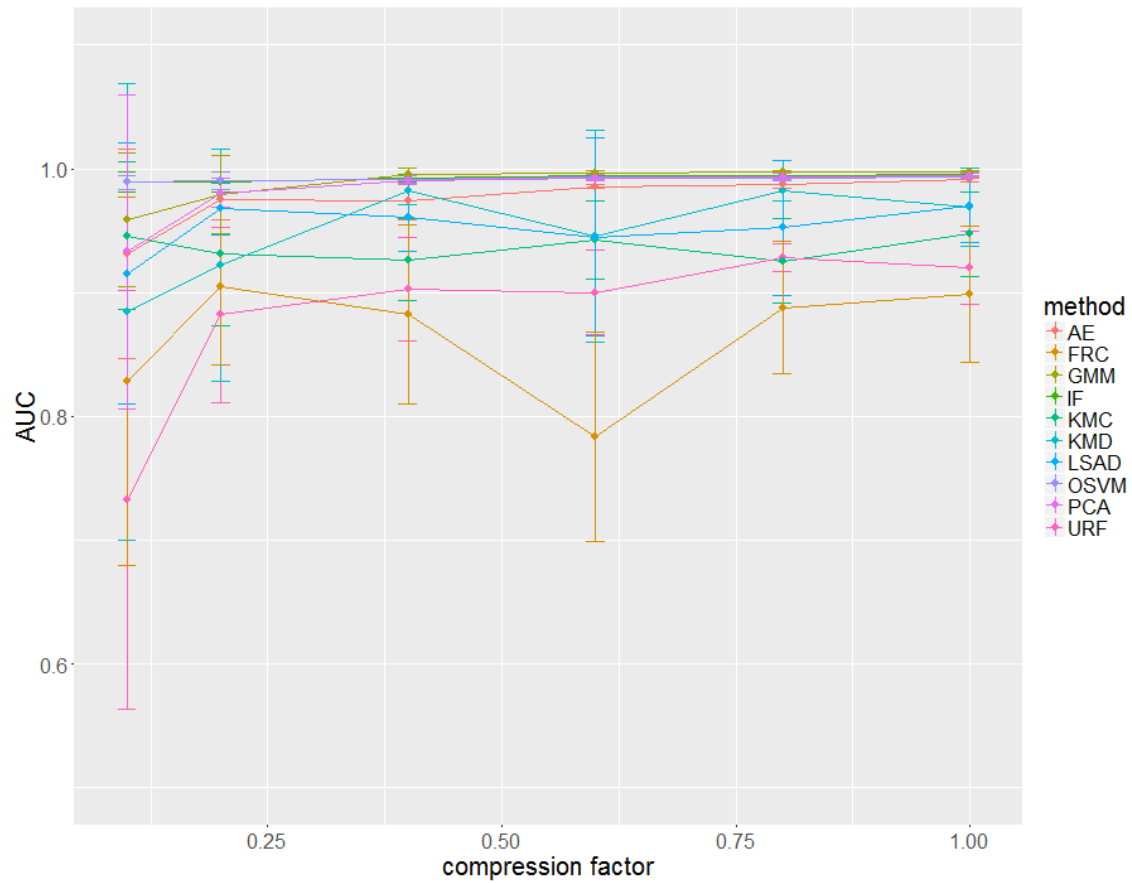


Figure 5.26: Performance of the different anomaly detection methods on the Network Intrusion Dataset with EE represented data, using $n=10\,000$ training samples and compression factors ranging from 0.1 to 1.0.

For this dataset all methods perform roughly constant with increasing compression factor for ≥ 0.2 .

EE Compression Curves - Higgs Challenge

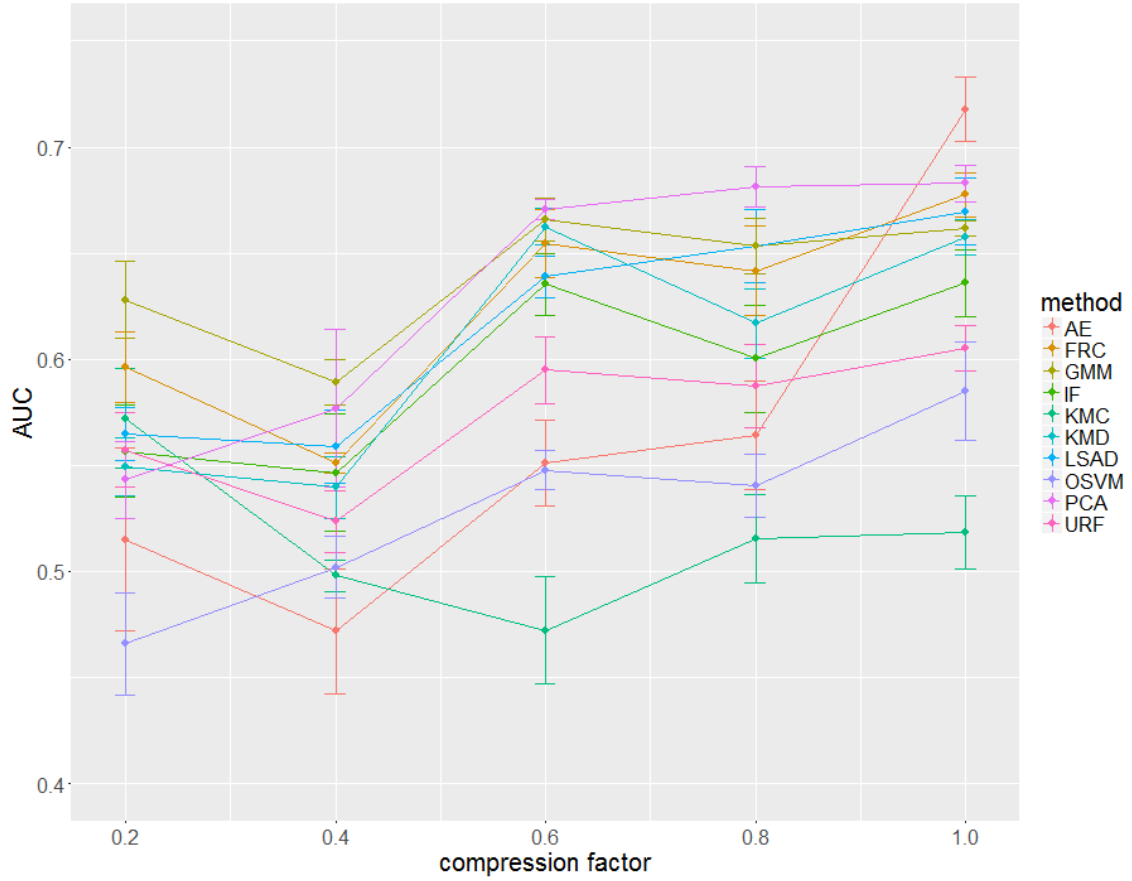


Figure 5.27: Performance of the different anomaly detection methods on the Higgs Challenge Dataset with EE represented data, using $n=10\,000$ training samples and compression factors ranging from 0.2 to 1.0.

For this dataset, all methods show roughly a monotonously increasing trend with increasing compression factor. The only exception builds KMC, which shows a "U-shaped" behaviour which is probably caused by noise effects.

To conclude one can say for the EE representation, that the methods FRaC, IF, KMD, LSAD and OSVM perform best with a factor of 1.0. Further the methods AE and PCA perform best for factor 0.8-1.0 and the methods GMM and KMC best for 0.4.

5.8 Robustness Curves

In this test we want to find out, how the different anomaly detection methods perform when noise features get added to the original features. In order to test that, we add numerical and categorical noise features to the original datasets and keep track of the performance, under increasing number of noise features. The added noise features we add, approximately have the same numerical/categorical ratio as the original features. The numerical noise features are standard normal distributed, the categorical noise features are categorical variables with 3 levels, each having equal probability.

Robustness Curve - Forest Cover Type

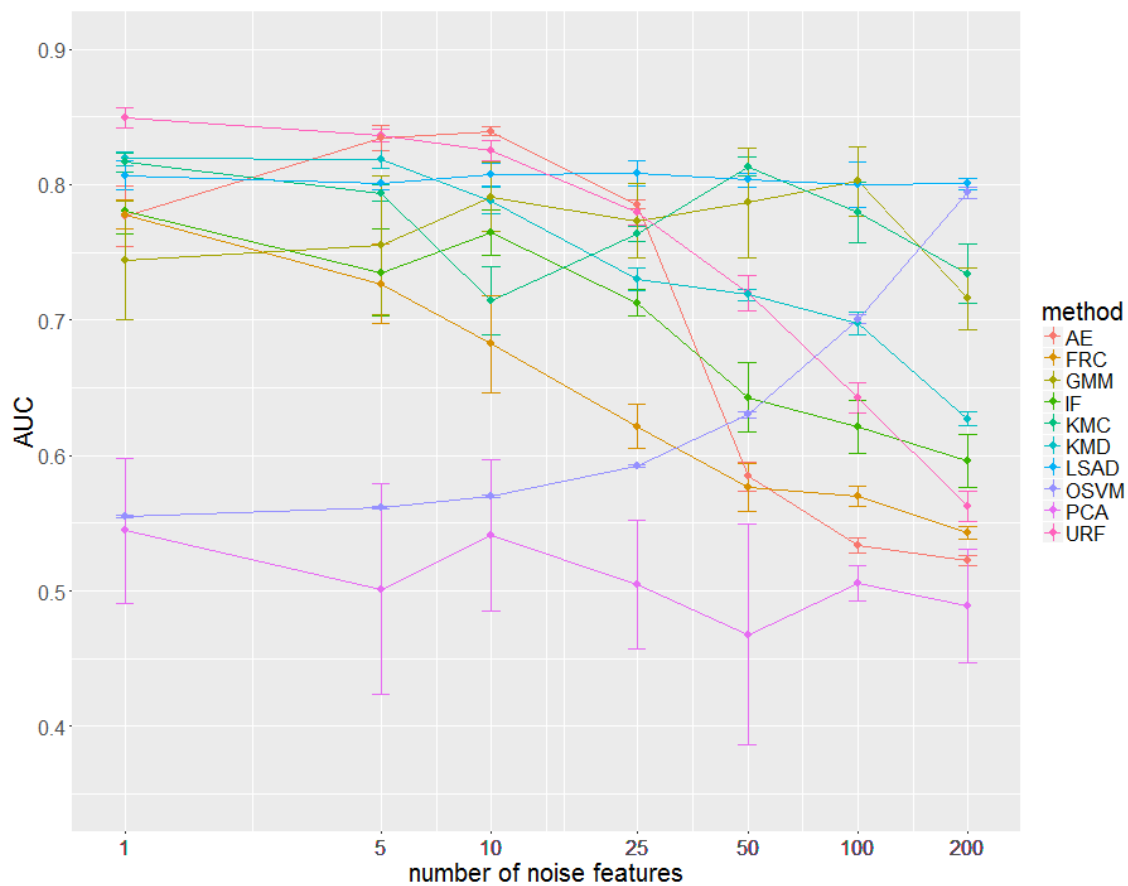


Figure 5.28: Performance of the different anomaly detection methods on the Forest Cover Type dataset with original data, using $n=10\,000$ training samples and a number of additional noise features 0, 1, 5, 10, 25, 50, 100, 200.

When investigating the behaviour of different methods under varying number of noise features, we can observe different behaviours. At first there are methods that show a strong decrease in performance with increasing number of noise features, such as KMD, URF, AE, IF and FRaC. Other methods are more stable with respect to the number noise features such as LSAD, GMM and KMC. Further there is the PCA method that does not perform significantly better than random, even for one noise feature and thus we don't

draw any conclusion from. The OSVM method shows an artefact that is confusing in the first place, because an increasing performance with addition of useless information doesn't make sense. The only explanation for this behaviour is, that the default parameters are more tailored towards the situation with many useless features and worse for the original setting, without any noise features.

Robustness Curve - Credit Card Fraud

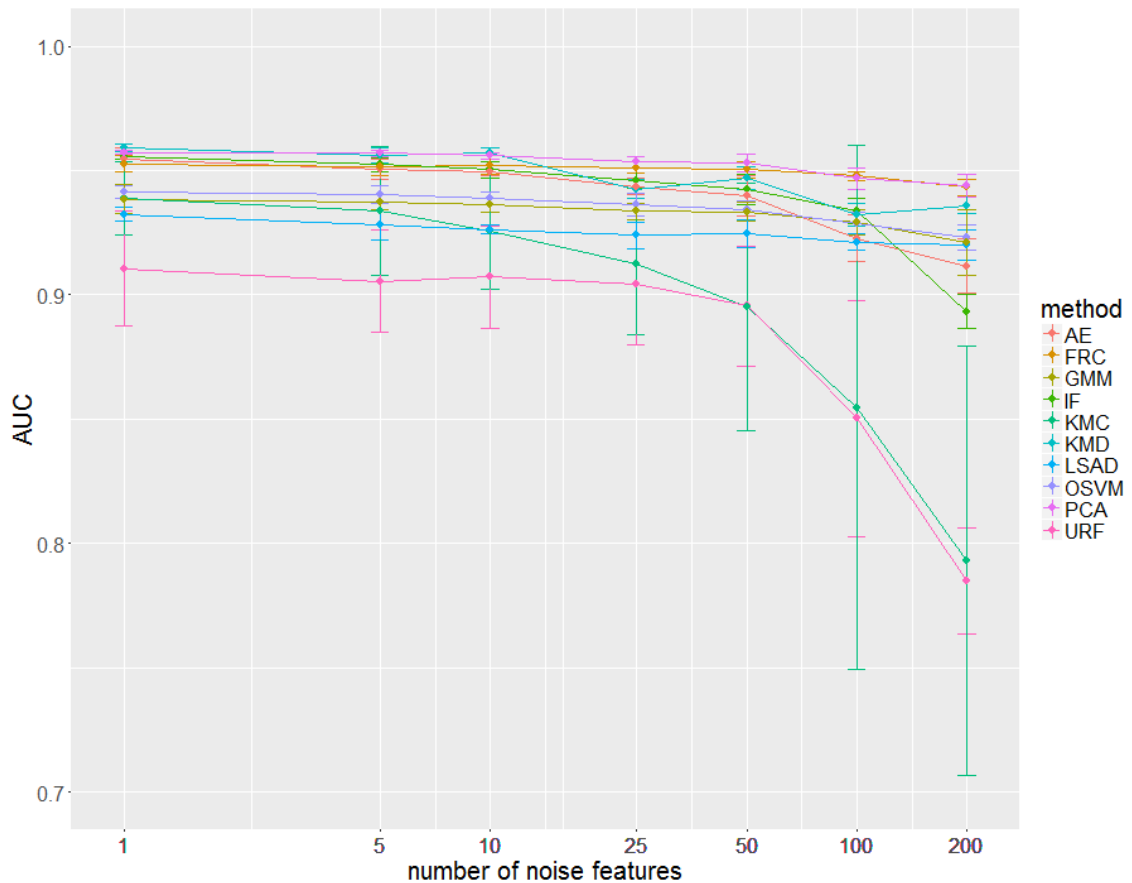


Figure 5.29: Performance of the different anomaly detection methods on the Credit Card Fraud dataset with original data, using $n=10\,000$ training samples and a number of additional noise features 0, 1, 5, 10, 25, 50, 100, 200.

The different methods worsen their performance at different rates, with increasing number of noise features. In decreasing order of decaying speed, the fastest decaying methods are URF, KMC, IF and AE. Very stable are the methods PCA, FRC, KMD, LSAD, OSVM, GMM.

Robustness Curve - Financial Data

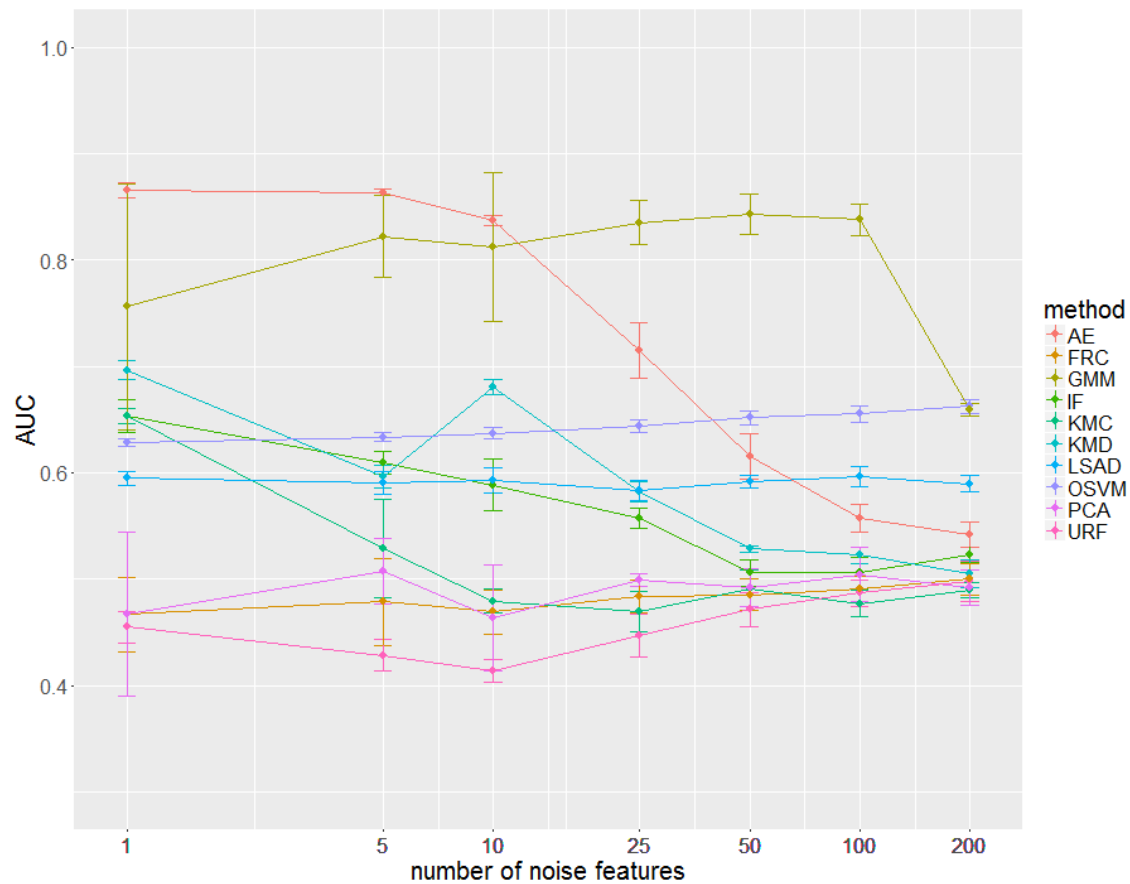


Figure 5.30: Performance of the different anomaly detection methods on the Financial Dataset with original data, using $n=10\,000$ training samples and a number of additional noise features 0, 1, 5, 10, 25, 50, 100, 200.

This dataset confirms the robustness of LSAD and OSVM against noise features. Further the are different speeds of performance decay for the other methods. A fast decrease in performance show the methods AE, KMC, IF, whereas the method GMM stays quite stable until 100 noise features.

Robustness Curve - Network Intrusion

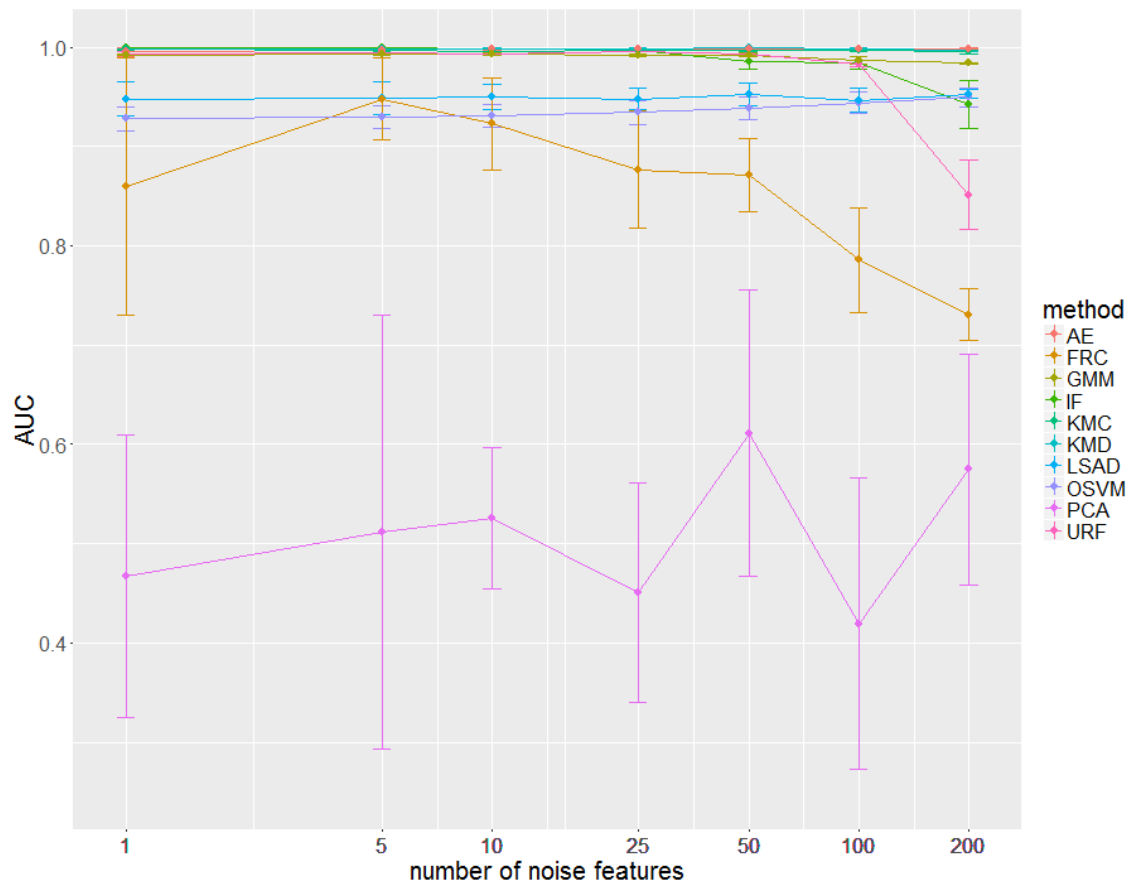


Figure 5.31: Performance of the different anomaly detection methods on the Network Intrusion dataset with original data, using $n=10\,000$ training samples and a number of additional noise features 0, 1, 5, 10, 25, 50, 100, 200.

This dataset shows again different speeds of performance decay for the different methods. As seen before, the LSAD and OSVM methods are robust against noise features. The PCA method doesn't perform better than random, so we can't say anything about it. The methods that show an observable performance decay are, in decreasing speed of decay, FRaC, URF, IF and GMM.

Robustness Curve - Higgs Challenge

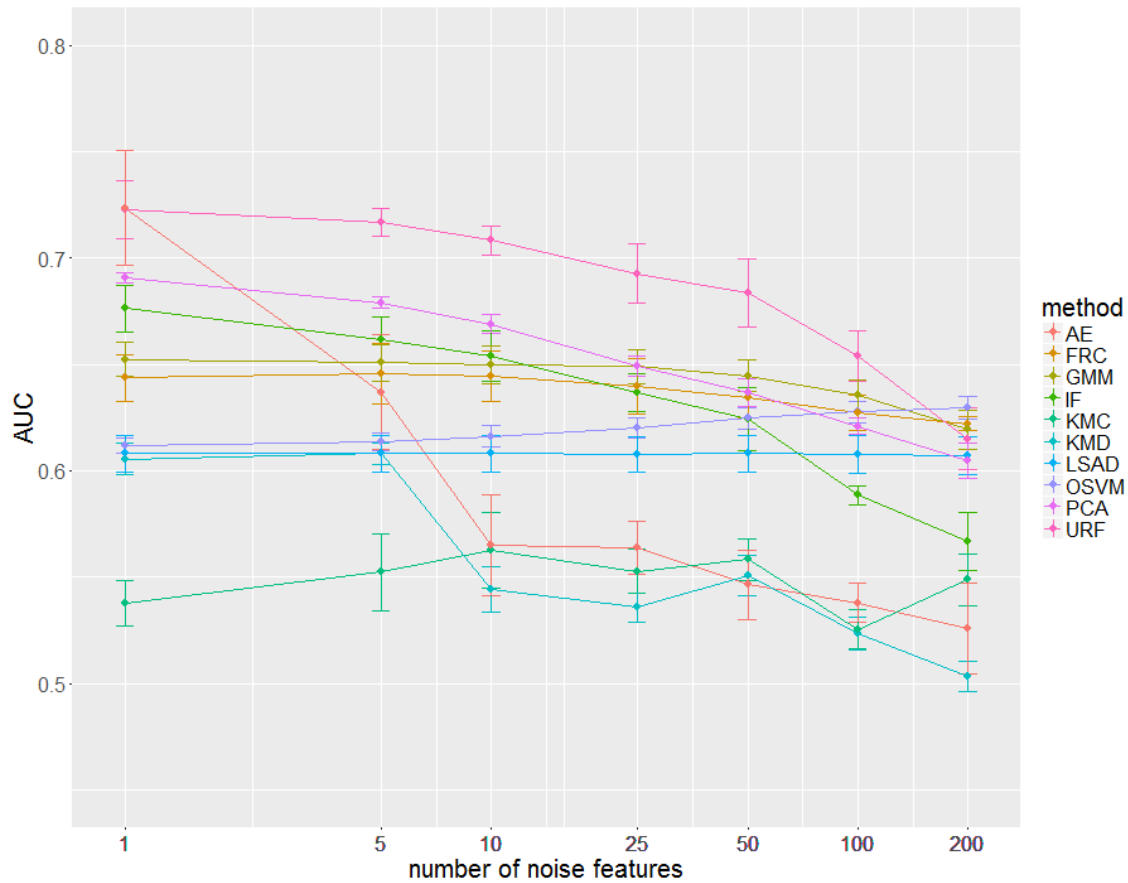


Figure 5.32: Performance of the different anomaly detection methods on the Higgs Challenge dataset with original data, using $n=10\,000$ training samples and a number of additional noise features 0, 1, 5, 10, 25, 50, 100, 200.

This dataset shows again different speeds of performance decrease for different methods. For example the methods LSAD, OSVM, FRaC and KMC are very stable with respect to the number of noise features. Other methods show a faster decreasing noise curve, such as PCA, AE, URF, GMM, IF and KMD.

Thus when considering the results of all the datasets, we can conclude that when a lot of noise features are expected, the methods LSAD, OSVM or GMM are strongly favourable compared to the other methods. But caution is necessary, because here we tested robustness against a very specific type of noise, which may not be representable and too simple for noise features occurring in real datasets.

5.9 Visualization in 2D

Unfortunately its very difficult to visualize how the scoring functions of the different anomaly detection methods behave in high dimensional, mixed-type data spaces. Only in the case of real valued, two dimensional datasets we can make a useful visualization. Thus we will use the butterfly dataset (see Figure 3.1), to investigate the score distribution on this dataset.

In a first step, we fit the algorithms to the butterfly data. Then we compute the anomaly scores of the training data points as well as for each point on a 100×100 grid around the dataset. Then we plot a scatter of the data points, with colour indicating how anomalous they are considered by the anomaly detection method. Increasing belief in degree of anomaly is indicated by the colouring of the points ranging from blue over green to red. Further we add a colour gradient (with 20 levels) to the areas around the points to indicate, how anomalous the new points on the grid around the original points would be considered by the anomaly detection methods. Areas in dark blue stand for a low anomaly scores and areas in light blue stand for a high anomaly scores.

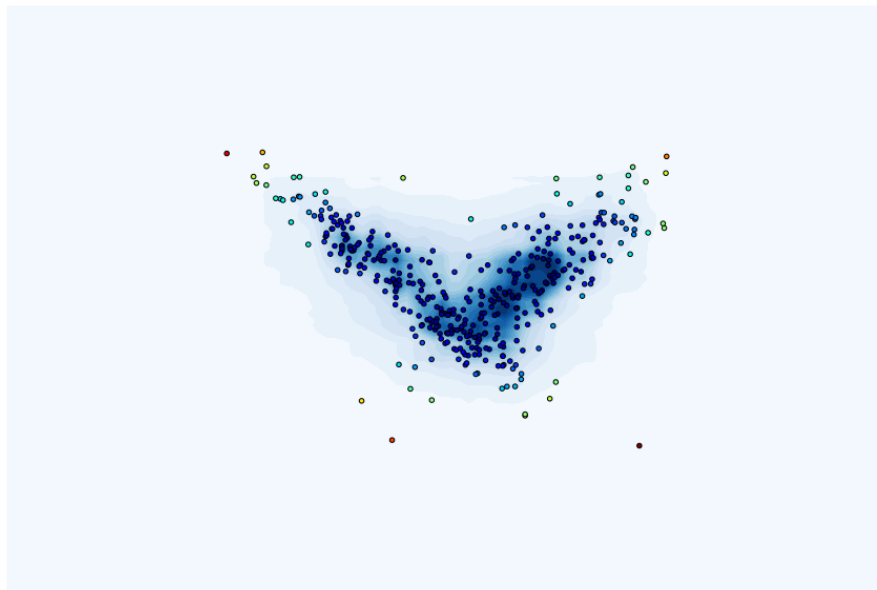


Figure 5.33: This plot shows how the Isolation Forest (rank) score behaves for the different regions around the butterfly data points, using a colour gradient.



Figure 5.34: This plot shows how the Unsupervised Random Forest (rank) score behaves for the different regions around the butterfly data points, using a colour gradient.

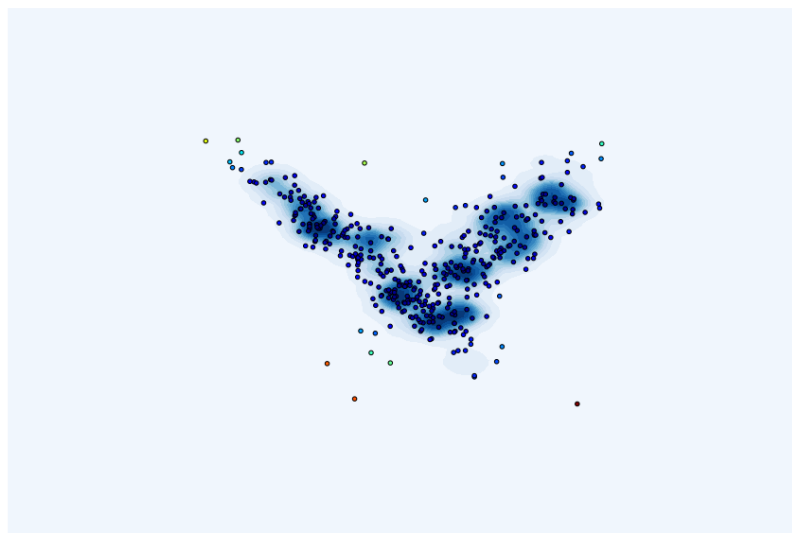


Figure 5.35: This plot shows how the k -means distance (rank) score behaves for the different regions around the butterfly data points, using a colour gradient.

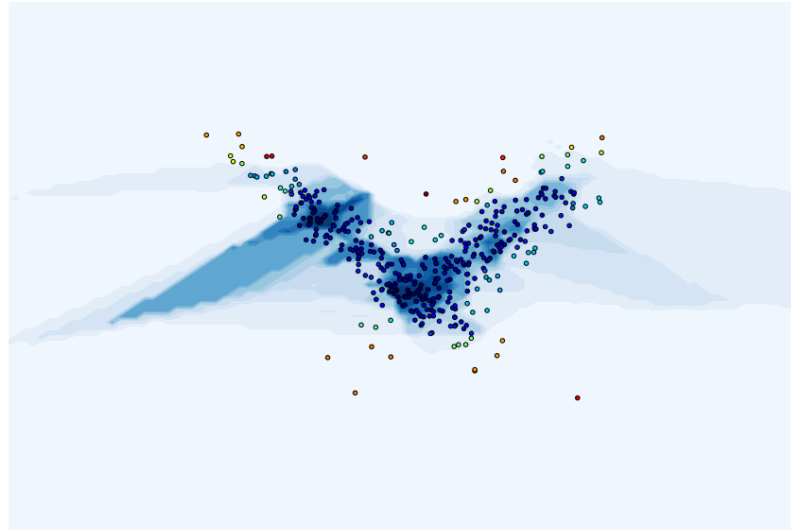


Figure 5.36: This plot shows how the k -means cluster-size (rank) score behaves for the different regions around the butterfly data points, using a colour gradient.

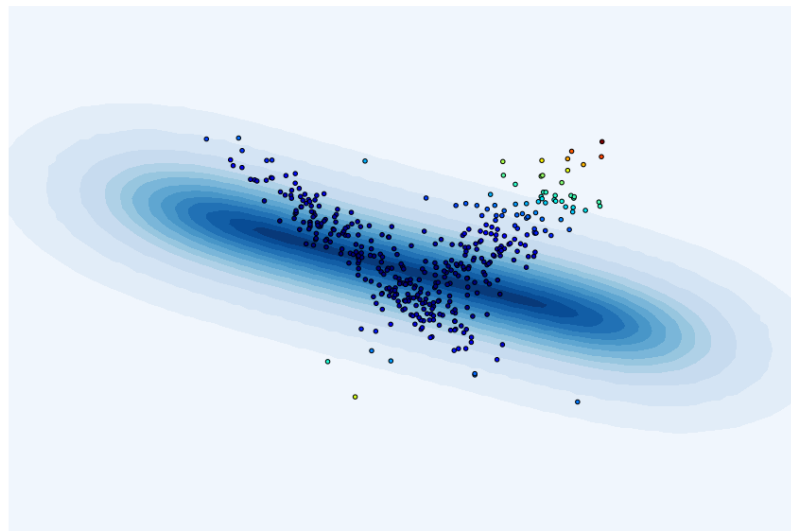


Figure 5.37: This plot shows how the Autoencoder (rank) score behaves for the different regions around the butterfly data points, using a colour gradient.

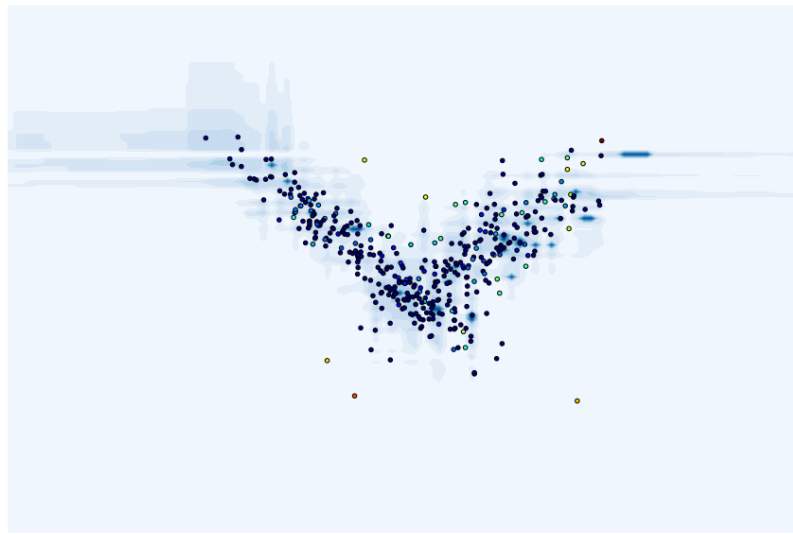


Figure 5.38: This plot shows how the Feature Regression and Classification (rank) score behaves for the different regions around the butterfly data points, using a colour gradient.

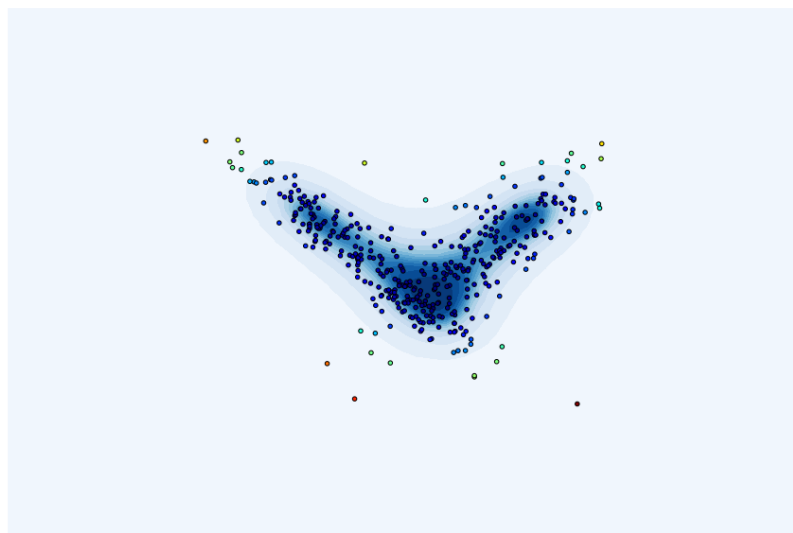


Figure 5.39: This plot shows how the one-class SVM (rank) score behaves for the different regions around the butterfly data points, using a colour gradient.

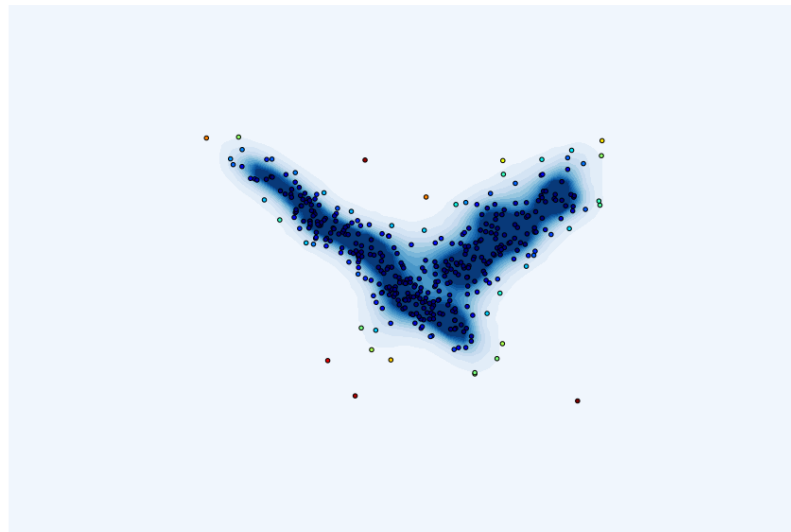


Figure 5.40: This plot shows how the Least Squares Anomaly Detection (rank) score behaves for the different regions around the butterfly data points, using a colour gradient.

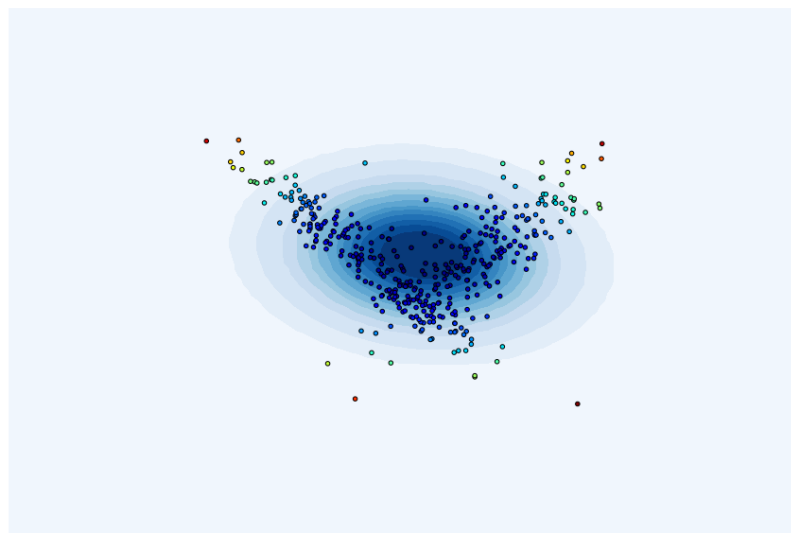


Figure 5.41: This plot shows how the Probabilistic PCA (rank) score behaves for the different regions around the butterfly data points, using a colour gradient.

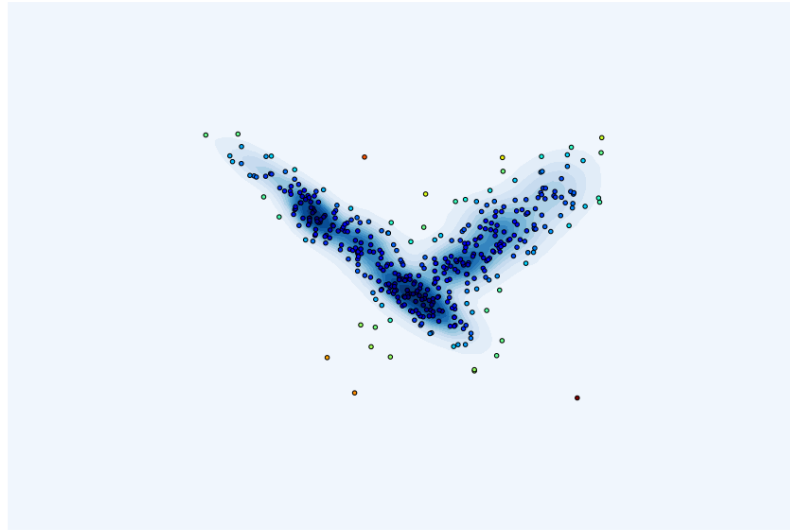


Figure 5.42: This plot shows how the Gaussian mixture model (rank) score behaves for the different regions around the butterfly data points, using a colour gradient.

We can conclude from the Figures 5.33 - 5.42 that some anomaly detection methods show intuitively very reasonable areas of high and low anomaly score values such as IF (Figure 5.33), KMD (Figure 5.35), OSVM (Figure 5.39), LSAD (Figure 5.40) or GMM (Figure 5.42). On the contrary, there are methods that represent the data with insufficient complexity, such as AE (Figure 5.37) and PCA (Figure 5.41) or show strange, unfortunate artefacts, such as URF (Figure 5.34), KMC (Figure 5.42) or FRaC (Figure 5.38).

Note that this pictures should be an opportunity to visualize the distribution of the scores and not be an argument to promote or discard certain algorithms. One has to keep in mind that the anomaly detection tasks we seek to solve, are typically mixed-type and have a lot more dimensions than just the two in this toy dataset. In addition, typical datasets for this task have of a lot more samples available for training. Thus it can well be the case, that methods perform bad on such two dimensional toy datasets, but are superior to other methods on real datasets of the before mentioned form.

Chapter 6

Summary

This chapter presents the main results of the experiments and describes, what insights they gave us. Further the chapter covers the problems and limitations of the work, to give an adequate impression of its usefulness. In addition it points out, where the road of the field may be heading and it states intentions about the possible areas of application of the presented work. Then, the end of the chapter completes the thesis, by embedding the topic of anomaly detection in a greater context.

6.1 Conclusions

As expected, there is no method outperforming all the other methods on all datasets and representations, as we can see in the experiments in chapter 5. Nonetheless there are some trends visible.

The AE method often shows good results on the original data, probably due to its compelling way of handling categorical variables. In contrast, the methods PCA, OSVM and FRaC often deliver poor performances on the original datasets. In addition, the methods IF and GMM always delivers moderate performance and thus are potential candidates for a default, out of the box method. If the dataset available for training is really small, e.g. 10 – 100 samples, the tests show, that one of the methods KMD, GMM or AE should be chosen.

Further we can observe specific "representation+anomaly detector"-combinations that steadily deliver good results.

As an example, the combination of PCA representation with compression factor 0.6 and GMM (PCA0.6+GMM) often perform well. Other interesting combinations are PCA0.8+IF and PCA0.8+AE.

Useful combinations of the autoencoder representation and anomaly detection methods are AE1.0+FRaC, AE0.8+AE, AE0.4+GMM and AE0.8+PCA.

Some potentially useful combinations of the Entity Embedding representation and anomaly detection methods are EE1.0+GMM, EE0.8+PCA, EE0.6+IF and EE1.0+FRaC.

When considering the behaviour of all methods over all datasets, we can detect certain pattern. The method GMM performs generally better for smaller compression factors (0.2 – 0.6). Further the PCA anomaly detection method tends to perform well on com-

pression factor 0.8, independent of which representation has been used. Other methods like FRaC, IF and URF generally prefer a big representation, i.e. a compression factor of 1.0. Another interesting method is LSAD, which performs often astoundingly constant with varying compression factor.

When it comes to robustness against additional noise features, our tests show that the most stable methods are the conceptually closely related methods OSVM and LSAD. Further GMM shows also a very stable behaviour. Probably not useful with many noise features are in contrast the methods URF, IF, AE, KMD and PCA.

6.2 Problems and Limitations

Anomaly detection methods

Certainly one the biggest problems, when doing tests with anomaly detection methods, is the choice of the parameters. Almost all methods, covered in this thesis, have many parameters to specify and in addition, the results are often strongly depending on their choice. Therefore, it can be hard to compare them, because for almost each dataset one can find parameters for which a method performs comparatively good or bad with respect to the other methods. To solve that problem, we specified default parameters that frequently perform well on different datasets. Nevertheless this choice is very ad hoc and further tests need to be done, to get an intuition how the choice of parameters affect the fitted model in different circumstances. Thus the experiments should be taken with a pinch of salt and one has keep in mind, that the performances of certain methods may drastically change with different default parameters.

Data

Another big problem when investigating the performance of anomaly detection methods, is to get useful datasets for anomaly detection. We solved this problem with the same trick as used in other papers (e.g. [Noto et al. \(2012\)](#)), by using a classification dataset to produce an anomaly detection training and test datasets. Since different classes generally show different behaviours of the predictor variables, one can deduce that their distributions differ. This is used to generate anomaly datasets, by specifying observations of one class as anomalies and the rest as normal samples. Of course this makes the implicit assumption, that these produced datasets are representative for typical anomaly detection tasks, which may be contested. Nonetheless this way of producing anomaly datasets for testing is probably better than using synthetically generated data, because it often fails to represent the typical complexity of real datasets.

To investigate the performance of the methods with respect to each other, we used five different datasets from multiple application domains. Even in this five datasets we met an immense variation of behaviours and thus, it is difficult or even impossible to deduce general rules of behaviour for the different methods. Especially the small number of five "sample datasets" does not allow for generalization and it is not clear how representative they are for general anomaly detection datasets.

In a real life scenario, one would probably use the small amount of anomaly data already

observed in the test set and choose the best performing anomaly detection method. In addition one can catch a glimpse on what hyperparameter choices of the detection method are reasonable for this dataset. However, one has to be cautious and should not overdo it with optimization, because one can end up basically fitting a classifier. This is not the goal one wants to achieve, because one can lose the actual strength of the anomaly detection method, to detect new types of anomalies.

Representation

The choice of parameters problem mentioned before, gets exponentiated by the usage of representations. Even just the number of (representation, detection method)-pairs and their performances on the different test datasets, are hard to overlook. Let alone all the parameters to choose for the representation method, such as the compression factor, the number of epochs to train the neural network based representation methods, etc.

6.3 Future Work

There are numerous ways to extend the work in this thesis.

First, there are other interesting anomaly and representation methods, which we didn't try out. For example the Restricted Boltzmann Machine (RBM), originally designed for binary data, has recently been adapted to an interesting range of different data types such as numerical, binary, categorical and counts. The new method called Mixed-variate RBM (Mv.RBM) (Do et al., 2016) can be used for representation as well as directly for anomaly detection, using the free-energy as an outlier score. It has further been improved successfully by extending the Mv.RBM to a Mixed-variate Deep Belief Network (Mv.DBN) (Do et al., 2016) and for this reason, the approach of Mv.RBM's presents an interesting approach which should be tested.

Further there are other anomaly detection methods like Beta mixture model (BMM) (Bouguessa, 2015), SOM (Shahreza, Moazzami, Moshiri, and Delavar, 2011), ODMAD (Koufakou, Georgiopoulos, and Anagnostopoulos, 2008), GLM- t (Lu, Chen, Wang, and Lu, 2016) we didn't have time to try out. Similarly there are other representation models we didn't cover, such as Factor Analysis or Independent Component Analysis

In addition there are multiple ways to improve certain algorithms we discussed. For example for the FRaC method, we used only the random forest as a supervised learner but in the original paper they suggested an ensemble of models to improve the performance.

The experiments should be expanded further in a more structured and extensive manner. One way to do this, is to investigate how the performance of the anomaly detection methods (in combination with representations) behave under different ratios of categorical/numerical features or number of levels of the categorical variables. This could be done by testing the methods on an extensive collection of data sets and by leaving away of certain features or by testing on a synthetically generated dataset with the wanted properties.

Another interesting question is how one should deal with data, that contains missing values. The approach of blindfolded imputation of the missing values, using any imputation

method, may not be the best idea. This way of handling missing data does probably not work well for real life data, because often there is signal contained in the pattern of missing values, that can be exploited. Thus it probably makes sense, to impute the missing values in a way that the algorithms can distinguish them from the other values. One solution would be to introduce a "missing" level for categorical variables and to use a value outside of the feature range for missing values of numerical features.

Analogue to supervised learning methods, one could also be interested in ensembling diverse anomaly detection algorithms, to build a model combining the strengths of different anomaly ideas. Because of the different ranges of the scores, one cannot directly combine them. However, using the *rank score* we introduced in equation (2.4.0.1), one can for example take the mean or a weighted mean of rank scores of different anomaly detectors to combine them in a controlled manner.

Similar to supervised learning, one can also be interested in calculating a proper, unbiased estimate of the performance, similar to cross validation. This can for example be done by repeating the whole novelty detection pipeline (including representation) for f -times (e.g. $f = 5$), while in each iteration one uses a fraction of $\frac{f-1}{f}$ of the normal samples as training samples and $\frac{1}{f}$ of the normal samples together with $\frac{1}{f}$ of the anomalies as the test set.

Another interesting direction to extend the work, is to consider the problem of Outlier Detection, which has been described in section 2.3. The idea of Outlier Detection is essentially to do Novelty Detection with a contaminated training set, i.e. a training set that already contains a few anomalies. We can use exactly the same pipeline as implemented for the Novelty Detection task with the only difference, that we use a training set already containing some anomalies. The main task of the anomaly detection methods is to build a model, that captures well the main pattern but is robust enough, not to overfit to the anomalies. We implemented this scenario in the test program as well, so the interested reader may explore this task by himself.¹

6.4 Potential Applications

Potential applications of anomaly detection can be found in any situation, where the goal is to find data points, which are not following the majority of data, i.e. finding novelties or outliers.

One area where anomaly detection is already used, is *fraud detection*. As an example it can be used to detect credit card fraud, financial transaction fraud, insurance fraud or medical fraud. The reason why anomaly detection is more useful in this scenario than for example classification, is that often the availability of fraud samples is very restricted or not available at all. However, even if data is available, it is often not representative of all possible ways of fraud that occurred so far or occur in the future. The methods we investigated in this thesis are especially suited for these problems and this kind of data, because it is usually mixed-type data.

Another possible application could be in scientific research, in cases where the amount of data one wants to classify, is outgrowing the capacity of manual classification. This is the case in astronomy, where recent surveys of telescopes produce data that overgrow

¹Check the Appendix A for the source of the code and description of the test program.

any human capacity to classify. One wants to solve the problem of classifying the data as well finding novel yet still unknown pattern. To accomplish this, one could feed anomaly detection methods with the already classified data and use the anomaly detector to find interesting new objects, that do not match this already seen classes. This procedure could improve efficiency by leaving the "boring" examples to a classifier algorithm and let human capacity deal with the potentially "interesting", new and unknown objects.

Other applications for anomaly detection may include network intrusion detection, visual processing or textual anomaly detection.

6.5 The Big Picture

The current trends in technology and economy demand more than ever before, to build in data-driven "smartness" into applications, processes and decisions. Machine learning is the technique that fulfils that need. It enables us to distil knowledge from data, in form of implicit understanding of relations and structure.

This field has currently been overwhelmingly dominated by the well understood and historically old techniques of regression and classification. Then in the 90's and early 00's the (often unsupervised) data mining techniques entered the field of machine learning with great success.

In current years, the field got enriched by new emerging sub-fields, such as recommendation techniques, visual perception and natural language processing. The later two succeeding especially by using convolutional and recurrent neural networks. This techniques opened up a whole new world of applications, unimaginable even 20 years before, such as autonomous driving, toddler like understanding of pictures, intelligent chat bots, etc.

Other interesting advances include reinforcement learning, which allows an actor to learn from an unknown environment through interaction to maximize some reward. This technique enabled the program AlphaGo to beat the world best Go-player in a match in March 2016.

Further outlook includes techniques, such as generative adversarial networks, that may help adding some fantasy and imagination to the machine learning toolbox. Applications may for example include generating a photo-realistic picture of a caption describing the content of a picture.

Anomaly detection has its own place in this fast expanding toolbox of machine learning tools. As mentioned in the section before, we can for example build filters as a pre-step to supervised algorithms to check, if the input arises from the same distribution as the training data of the supervised learner. Other applications may want to automatically find new pattern in data. Thus Anomaly Detection is a collection of methods, that can extend the reach and capabilities of the machine learning field even further.

An important factor to make this methods, including anomaly detection, suitable for today's requirements, is to parallelize them. Parallelization makes them applicable in a "Big Data"-environment and allows to speed up the computation time, by using all resources available. Thus all the anomaly detection methods and representation models we presented in this thesis (with exception of OSVM), already use or potentially allow for parallelization. In addition all the neural network based approaches to anomaly detection

or representation and maybe also other methods, can potentially make use of GPU's using the right implementation, that can additionally considerable speed up the methods.

Bibliography

- Arthur, D. and S. Vassilvitskii (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027–1035. Society for Industrial and Applied Mathematics.
- Bengio, Y., A. Courville, and P. Vincent (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35(8), 1798–1828.
- Bishop, C. (2007). Pattern recognition and machine learning (information science and statistics), 1st edn. 2006. corr. 2nd printing edn. *Springer, New York*.
- Bouguessa, M. (2015). A practical outlier detection approach for mixed-attribute data. *Expert Systems with Applications* 42(22), 8637–8649.
- Chandola, V., A. Banerjee, and V. Kumar (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41(3), 15.
- Do, K., T. Tran, D. Phung, and S. Venkatesh (2016). Outlier detection on mixed-type data: An energy-based approach. In *Advanced Data Mining and Applications: 12th International Conference, ADMA 2016, Gold Coast, QLD, Australia, December 12-15, 2016, Proceedings 12*, pp. 111–125. Springer.
- Do, K., T. Tran, and S. Venkatesh (2016). Multilevel anomaly detection for mixed data. *arXiv preprint arXiv:1610.06249*.
- Duchi, J., E. Hazan, and Y. Singer (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12(Jul), 2121–2159.
- Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>.
- Guo, C. and F. Berkhahn (2016). Entity embeddings of categorical variables. *arXiv preprint arXiv:1604.06737*.
- Koufakou, A., M. Georgiopoulos, and G. C. Anagnostopoulos (2008). Detecting outliers in high-dimensional datasets with mixed attributes. In *DMIN*, pp. 427–433.
- Kwedlo, W. (2014). A parallel em algorithm for gaussian mixture models implemented on a numa system using openmp. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pp. 292–298. IEEE.
- Liaw, A. and M. Wiener (2002). Classification and regression by randomforest. *R news* 2(3), 18–22.

- Lichman, M. (2013). UCI machine learning repository.
- Liu, F. T., K. M. Ting, and Z.-H. Zhou (2008). Isolation forest. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pp. 413–422. IEEE.
- Lu, Y.-C., F. Chen, Y. Chen, and C.-T. Lu (2013). A generalized student-t based approach to mixed-type anomaly detection. In *AAAI*.
- Lu, Y.-C., F. Chen, Y. Wang, and C.-T. Lu (2016). Discovering anomalies on mixed-type data using a generalized student- t based approach. *IEEE Transactions on Knowledge and Data Engineering* 28(10), 2582–2595.
- Markou, M. and S. Singh (2003). Novelty detection: a review – part 1: statistical approaches. *Signal processing* 83(12), 2481–2497.
- Nesterov, Y. (1983). *A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$* , Volume 269.
- Noto, K., C. Brodley, and D. Slonim (2012). Frac: a feature-modeling approach for semi-supervised and unsupervised anomaly detection. *Data mining and knowledge discovery* 25(1), 109–133.
- Pimentel, M. A., D. A. Clifton, L. Clifton, and L. Tarassenko (2014). A review of novelty detection. *Signal Processing* 99, 215–249.
- Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural networks* 12(1), 145–151.
- Quinn, J. A. and M. Sugiyama (2014). A least-squares approach to anomaly detection in static and sequential data. *Pattern Recognition Letters* 40, 36–40.
- Shahreza, M. L., D. Moazzami, B. Moshiri, and M. Delavar (2011). Anomaly detection using a self-organizing map and particle swarm optimization. *Scientia Iranica* 18(6), 1460–1468.
- Srivastava, N., G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15(1), 1929–1958.
- Tax, D. M. J. (2001). *One-class classification*. Ph. D. thesis, TU Delft, Delft University of Technology.
- Tipping, M. E. and C. M. Bishop (1999). Mixtures of probabilistic principal component analyzers. *Neural computation* 11(2), 443–482.

Appendix A

Implementation - Methods and Testing Program

This appendix explains some details about the implementation used for the experiments in chapter 5. In the sections A.1 and A.2 we mention the important modules, that have been used for the different anomaly detection and representation methods respectively. Further the section A.3 describes the function, that performs the actual anomaly detection test and the section A.4 explains the different parts the test program. Section A.5 describes and explains the GUI, that was implemented to facilitate the interaction with the test function.

The code of the testing part is available in the Github repository: <https://github.com/Stacky2/AnomalyDetection>.

A.1 Details - Anomaly Detection Methods

This section briefly describes, which modules were used for the different anomaly detection methods.

Isolation Forest

We used the Isolation Forest implementation of the *sklearn* package, `sklearn.ensemble.IsolationForest`.

For details visit <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html>

k -Means Distance/Custer-Size Model

We used the k -means implementation of the python module of *H2O.ai*, `h2o.estimators.kmeans.H2OKMeansEstimator`.

For details visit <http://h2o-release.s3.amazonaws.com/h2o/rel-turing/10/docs-website/h2o-py/docs/modeling.html#h2okmeansestimator>

Unsupervised Random Forest

This method we implemented using the random forest classifier from the package *sklearn*, i.e. using the function `sklearn.ensemble.RandomForestClassifier`.

For details visit <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>.

Autoencoder

For the Autoencoder anomaly detection method we used the implementation of the python module of *H2O.ai*, `h2o.estimators.deeplearning.H2OAutoEncoderEstimator`.

For details visit <http://h2o-release.s3.amazonaws.com/h2o/rel-turing/10/docs-website/h2o-py/docs/modeling.html#h2oautoencoderestimator>.

Feature Regression and Classification

This method we implemented ourselves using again the random forest classifier and regressor from the *sklearn* package, `sklearn.ensemble.RandomForestClassifier` and `sklearn.ensemble.RandomForestRegressor`.

For details visit <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html> and <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>.

The original implementation of FRaC from the paper [Noto et al. \(2012\)](#) can be found under link <http://bcb.cs.tufts.edu/frac/>.

One-Class SVM

We used the one-class SVM implementation of the *sklearn* package, `sklearn.svm.OneClassSVM`.

For details visit <http://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html>

Least Squares Anomaly Detection Method

We used the implementation of the paper about Least Squares Anomaly Detection [Quinn and Sugiyama \(2014\)](#), which is available in python, `lsanomaly.LSanomaly`.

For details visit <http://air.ug/~jqinn/software/lsanomaly.html>

Probabilistic PCA

We used the PCA implementation of the *sklearn* package, `sklearn.decomposition.PCA`.

For details visit <http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>.

Gaussian Mixture Model

We used the one-class SVM implementation of the *sklearn* package, `sklearn.mixture.GaussianMixture`.

For details visit <http://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html#sklearn.mixture.GaussianMixture>

A.2 Details - Representation Methods

In this section We briefly describe which modules where used for the different representation methods.

PCA

We used the PCA implementation of the *sklearn* package, `sklearn.decomposition.PCA`.

For details visit <http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>.

Autoencoder

For the Autoencoder anomaly representation method we used the implementation of the python module of *H2O.ai*, `h2o.estimators.deeplearning.H2OAutoEncoderEstimator`.

For details visit <http://h2o-release.s3.amazonaws.com/h2o/rel-turing/10/docs-website/h2o-py/docs/modeling.html#h2oautoencoderestimator>.

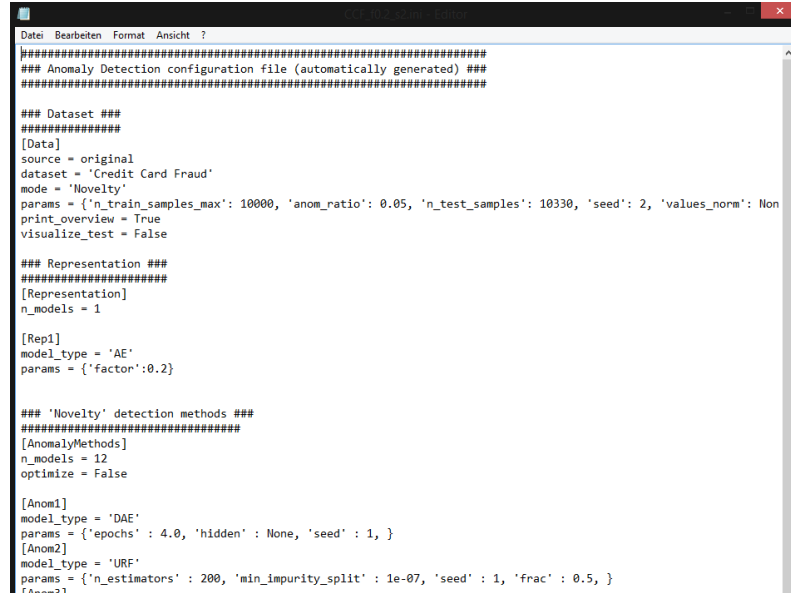
Entity Embedding

For the Autoencoder anomaly detection method we used the implementation of the python module of *H2O.ai*, `h2o.estimators.deeplearning.H2ODeepLearningEstimator`.

For details visit <http://h2o-release.s3.amazonaws.com/h2o/rel-turing/10/docs-website/h2o-py/docs/modeling.html#h2odeeplearningestimator>.

A.3 Test Function

For the actual test of the methods on a dataset we wrote the test function *execute_job* which is placed in the file *TestAnomalyMethods.py* in the program folder. The function *execute_job(config_name)* executes the anomaly detection test, using the parameters described in the configuration file *[config_name].ini*, which needs to be located in the *config*-folder of the program.



```

Datei Bearbeiten Format Ansicht ?
#####
### Anomaly Detection configuration file (automatically generated) ###
#####

### Dataset ###
#####
[Data]
source = original
dataset = 'Credit Card Fraud'
mode = 'Novelty'
params = {'n_train_samples_max': 10000, 'anom_ratio': 0.05, 'n_test_samples': 10330, 'seed': 2, 'values_norm': Non
print_overview = True
visualize_test = False

### Representation ###
#####
[Representation]
n_models = 1

[Rep1]
model_type = 'AE'
params = {'factor':0.2}

### 'Novelty' detection methods ###
#####
[AnomalyMethods]
n_models = 12
optimize = False

[Anom1]
model_type = 'DAE'
params = {'epochs' : 4.0, 'hidden' : None, 'seed' : 1, }
[Anom2]
model_type = 'URF'
params = {'n_estimators' : 200, 'min_impurity_split' : 1e-07, 'seed' : 1, 'frac' : 0.5, }
[Anom3]

```

Figure A.1: This figure shows an example of a configuration file, used to perform an anomaly detection test.

These parameters are then used to execute the test. Missing specifications such as parameters for the detection methods or data set parameters, are automatically chosen by default by the function *execute_job*. These default parameters themselves are saved as well in configuration files *config.ini* in the corresponding folders (e.g. in the folders *AnomalyModels* or *data*). After a test has been finished, the function saves a log-file with the results in a text document in the folder *Log*.

A.4 Test Program

The complete test program is contained in the program folder and consists of multiple files and folders depending on one another. The main part of the test program is the test function *execute_job* contained in the *TestAnomalyMethods.py* file, as described in the last section. There are a multitude of different folders containing different parts of the program:

- *AnomalyModels*: Contains the python file *AnomalyModels.py* which contains the classes for the different anomaly models and a *config.ini* file that contains the default parameters for the anomaly detection algorithms.
- *data*: Contains folders with the names of the data sets. In each of these folders there is a *data.csv* file that contains the corresponding data set. Further the folder *data*

contains a *config.ini* file, in which important information about the different datasets (eg. name of the label column: "label_col" or a list of the names of the categorical features: "fac_cols") is stored.

- *DataGeneration*: Contains a python file *DataGeneration.py*, that contains the classes for the data models with functions that allow to simulate data for the anomaly detection tasks. Further the folder contains a *config.ini* file which contains the details for the data simulation of each data set.
- *AnomalyDataSet*: Contains a python file *AnomalyDataSet.py*, that contains a class for anomaly detection datasets with different functions that allow for example to generate train-test-splits.
- *Log*: This is the folder where the output files get saved, after the test method *execute_job* has been executed.
- *OptimizeParameters*: Contains a python file *OptimizeParameters.py*, that handles the optimization of the hyper parameters of the anomaly models. Further there is a *config.ini* file in the folder, that contains the parameters for the Bayesian optimization algorithm as well as the ranges, in which the optimization algorithm should optimize the parameters.
- *RepresentationModels*: Contains the python file *RepresentationModels.py*, which contains the classes of the representation models.
- *Visualization*: Contains the python file *Visualization.py*, that contains the functions to visualize data.

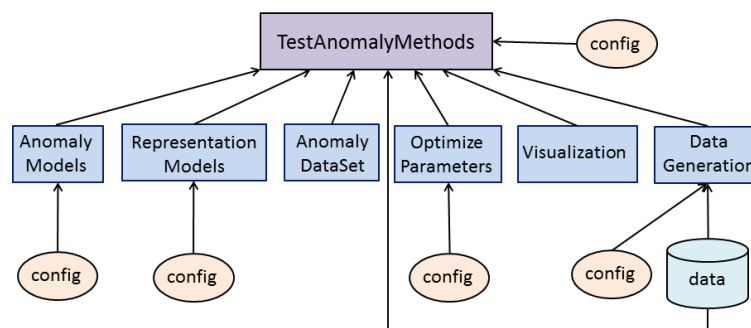


Figure A.2: This schema shows the dependencies of the files in the program.

A.5 GUI

To simplify the usage of the test function, which requires distinctive knowledge the different configuration variables, we implemented a little GUI to overcome this problem. Further the GUI was essential for the experiments, because we had to organize the test environment such that we could process whole batches of tests at once to produce the results for the numerous plots in chapter 5.

For example the test run for the Figure 5.8 in the experiments chapter consisted of 30 config files (6 train sets sizes, 5 split seeds) that had to be run, each testing 10 different methods which amounts to a number of 300 individual tests for this single plot.

For the tests we used a desktop computer with an "Intel(R) Core(TM) i7-4820K CPU @ 3.70GHz" CPU and 16,0 GB RAM and using all methods with maximal number of supported cores. For the different plots in chapter 5, the tests took between 2 and 7.5 hours.

When starting the python file *GUI.py*, a window opens up that can be used for performing tests. On can specify the different parameters of the test, in the corresponding tabs. Clicking the "Run!"-button in the "Run" tab triggers the program to collect the information from the different tabs and to form a configuration file for the test. This file gets saved in the *configs*-folder and then the test is executed with the *execute_job* function mentioned (explained in section A.3).

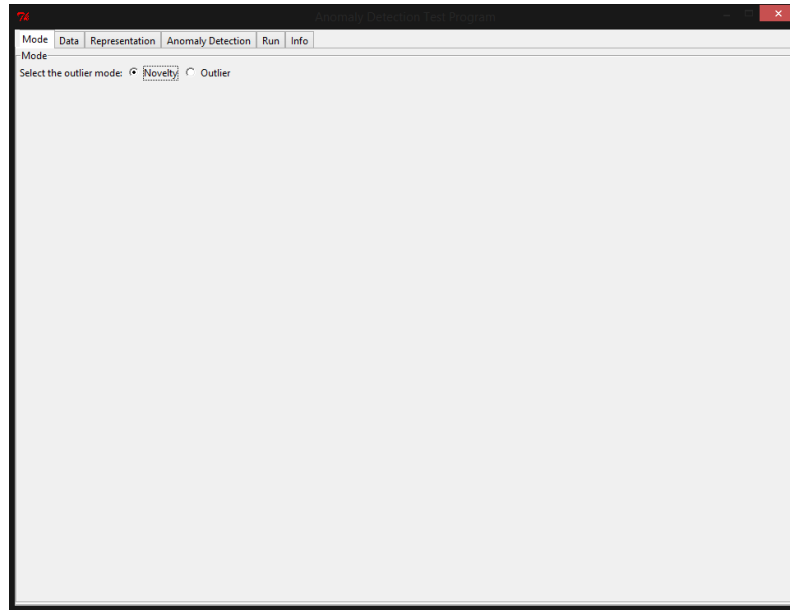


Figure A.3: This picture shows the mode tab of the GUI, in which we can specify if we want to perform a Novelty or an Outlier Detection test. Note that every test done in this thesis was about Novelty Detection.

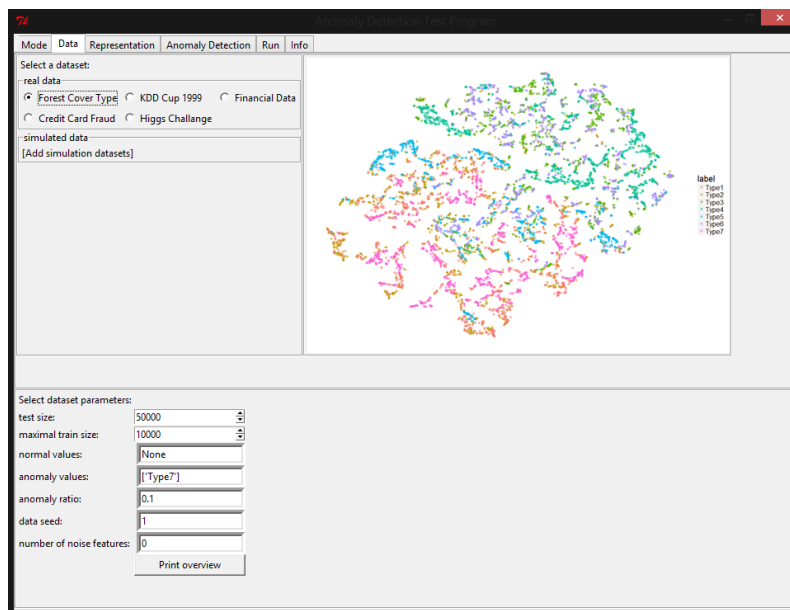


Figure A.4: This picture shows the data tab of the GUI, in which we can specify the dataset which should be used for the test. Further we can specify the parameters concerning the datasets.

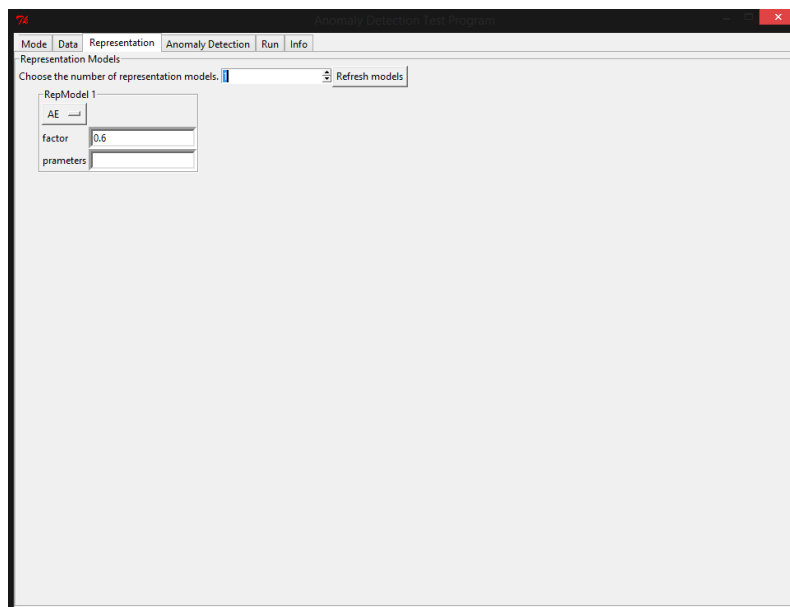


Figure A.5: This picture shows the representation tab of the GUI, that is used to specify the representation used for the data. If no representation should be used, choose the ID (Identity) model.

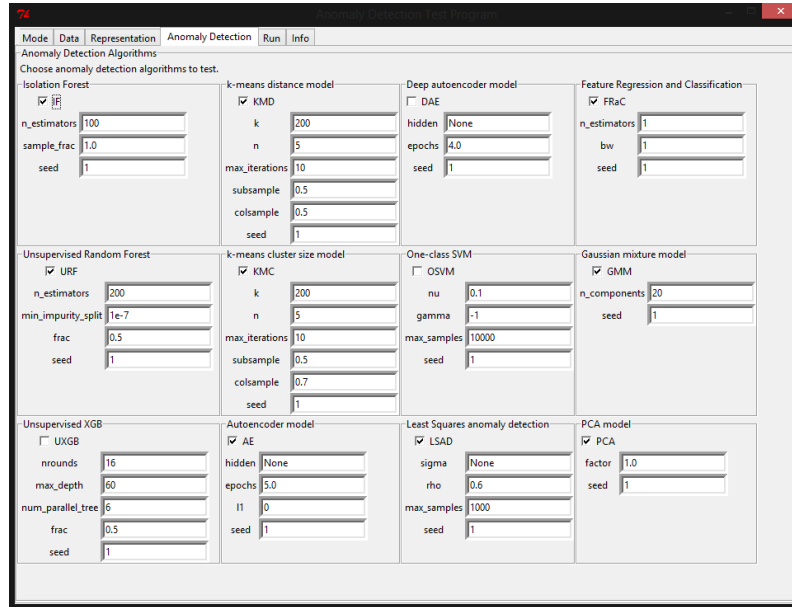


Figure A.6: This picture shows the anomaly method selection tab of the GUI. It is used to select the anomaly detection methods used for the test as well as their hyperparameter values.

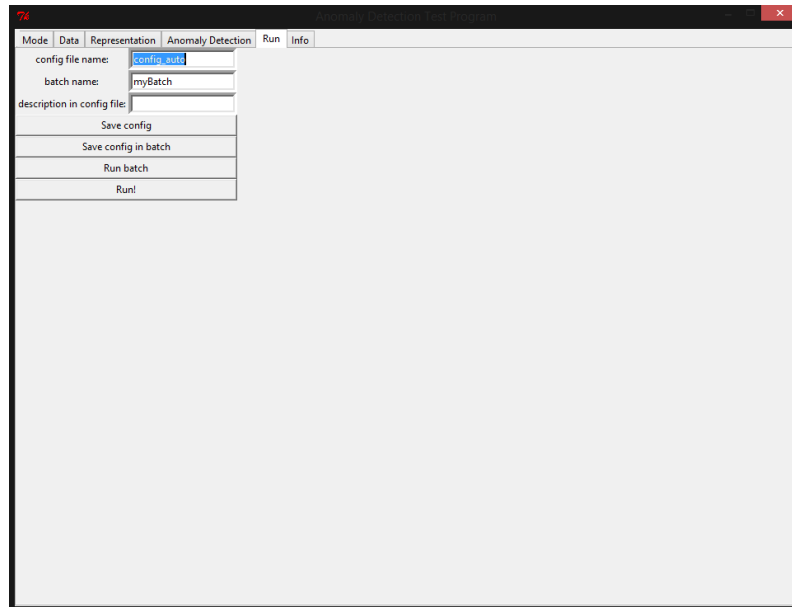


Figure A.7: This picture shows the Run section of the GUI, which is mainly used to start the test process. Additionally it allows to change the name of the automatically generated configuration file, to save configuration files in a batch (sub-folder of *config*-folder containing multiple config-files) and to execute all config-files in one batch. If a batch run is performed, the results are saved in the *results*-folder in form of a excel sheet with the corresponding performances of the methods.

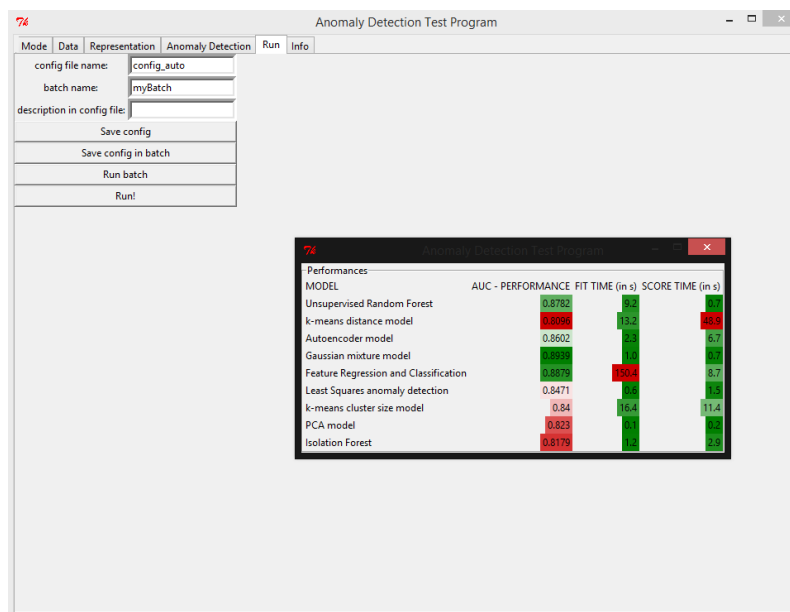


Figure A.8: This picture shows the output window of the GUI, which displays the results of the test run, after it has been started by the "Run!" button on the "Run" tab.

Declaration of Originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor .

Title of work (in block letters):

A comparison of Mixed-Type
Anomaly Detection Methods

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):

Mauchle

First name(s):

Mathias

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the **Citation etiquette** information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work .
- I am aware that the work may be screened electronically for plagiarism.
- I have understood and followed the guidelines in the document *Scientific Works in Mathematics*.

Place, date:

7 May 2017

Signature(s):

Mathias Mauchle

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.