Contents

[Notation 2](#_Toc361590701)

[1 Introduction 3](#_Toc361590702)

[1.1 Condition monitoring 3](#_Toc361590703)

[1.2 Overview of this thesis 5](#_Toc361590704)

[2 Feature Extraction 6](#_Toc361590705)

[2.1 Mel Frequency Cepstral Coefficients 6](#_Toc361590706)

[2.2 Multi Fractal Dimensions 8](#_Toc361590707)

[2.3 Kurtosis 9](#_Toc361590708)

[3 Semi-supervised classification 10](#_Toc361590709)

[3.1 Support Vector Data Description 13](#_Toc361590710)

[3.2 K-Center 16](#_Toc361590711)

[3.3 K-Means Data Description 17](#_Toc361590712)

[3.4 Nearest Neighbor Data Description 18](#_Toc361590713)

[3.5 Parzen Window Data Description 19](#_Toc361590714)

[3.6 Self Organizing Maps 20](#_Toc361590715)

[3.7 Random forest 22](#_Toc361590716)

[3.7.1 Variable importance 23](#_Toc361590717)

[3.7.2 Proximity 24](#_Toc361590718)

[3.7.3 Class prototypes 24](#_Toc361590719)

[3.7.4 Outlier measure 24](#_Toc361590720)

[3.7.5 Unsupervised training 25](#_Toc361590721)

[3.7.6 Semi-supervised classification 25](#_Toc361590722)

[4 Experiments 26](#_Toc361590723)

[4.1 Roller Bearing Datasets 27](#_Toc361590724)

[4.2 Feature Extraction 29](#_Toc361590725)

[4.2.1 Mel Frequency Cepstral Coefficients 29](#_Toc361590726)

[4.2.2 Higuchi Fractal Dimensions 30](#_Toc361590727)

[4.3 Random Forest Data Description 31](#_Toc361590728)

[4.4 Classification and Evaluation 33](#_Toc361590729)

[4.4.1 Performance evaluation 33](#_Toc361590730)

[4.4.2 Feature reduction experiment 35](#_Toc361590731)

[5 Conclusions 36](#_Toc361590732)

[6 References 38](#_Toc361590733)

# Notation

# Introduction

## Condition monitoring

*Condition monitoring* is the monitoring of a system “by studying certain selected parameters in such a way, that significant changes of those parameters are related to a developing failure” (Marwala, 2012). In manufacturing, condition monitoring can help to reduce operation costs significantly by preventing machine failure and supporting a condition based maintenance approach (Long). In other areas such as aerospace or construction, condition Monitoring can play a crucial role in the prevention of severe accidents.

Figure 1 illustrates a general *condition monitoring* framework (Marwala, 2012).



Fig. 1 Condition Monitoring Framework

A *data acquisition* devise comprises sensors measuring significant machine parameters and devices such as D/A-converters, which transform the acquired data into a signal suitable for further processing. Typical examples of sensors include accelerometers, thermometers or strain gauges. The outputs of data acquisition devices are usually discrete time signals, which can be represented by a set of tuples of the form

Where *N* denotes the number of pairs in the signal sequence and represents a sensor value such as the amplitude recorded at time .

Purpose of the data analysis step is the extraction of suitable attribute-value pairs – or *features* - from time signals, which can be used as inputs to decision algorithms. Data analysis techniques can roughly be grouped into three categories:

* *Time domain* methods, measuring aspects of the raw time signal directly (e.g. *kurtosis* or *fractal dimensions,* )
* *Frequency domain* transformation (*Fourier transform*)
* *Time-frequency domain* transformations, such as the *wavelet transform*, the *short time Fourier* transform or the *cepstrum* analysis

Outputs of the *Data Analysis* step are usually *feature* vectors of the form

where *n* denotes a specific data vector in a sequence of *N* vectors, ,…, are the particular *feature* values (*kurtosis*, *fractal dimension etc.*), and *D* is the *feature* set dimension.

The number of *features* *D*, can have a significant effect on the performance of classification algorithms, of which many are susceptible to the *curse of dimensionality* (Bishop, 2009). Several *feature reduction* techniques exist, to select the features which are best suited to distinguish machine conditions. By far the most popular of these methods is the *principal component* analysis.

A decision making device can be any method capable of estimating machine conditions from feature vectors. The available methods include statistical algorithms, machine learning and pattern recognition techniques or combinations of several methods, also known as *Batch* or *Boosting* algorithms (Marwala, 2012).

An important criterion for the selection of suitable decision techniques is the availability of *data samples* - also known as *training objects* - of the form

(1.1)

Where is the nth *feature* vector in a set of *N* sample vectors and denotes a known machine condition linked to the values of . Techniques utilizing sample data to construct a model of machine conditions are known as *supervised* methods. The term *supervised*,usually refers to sample data sets containing representative data samples for all known machine conditions. If sample data is available for only one condition, *semi-supervised* techniques can be used to train models which are able to distinguish the known condition from any other. *Semi-supervised* scenarios typically occur if it’s either too expensive or impossible to obtain sample data covering all possible conditions. In the *condition monitoring* of machines,data representing normal conditions can often be obtained easily during machine runtime, whereas the acquisition of data representing fault conditions requires the destruction of a machine in all possible ways (Myers, Japkowicz, & Gluck, 1995). In such scenarios, the normal samples can be used to train *semi-supervised* models which are able to detect anomalies during machine runtime. This principle is also known as *outlier* or *novelty detection* (Banerjee, Chandola, & Kumar, 2007).

If no sample data is available, *unsupervised methods* can be used to detect meaningful patterns in datasets. *Unsupervised* methods include clustering based approaches such as *hierarchical clustering (HCL)* (Carrascal, Diez, & Azpeitia, 2009)or statistical techniques.

The final step in a *condition monitoring*, the *condition diagnosis*, involves all conclusions which can be based on the results of the previous steps. Typical problems in this context are the estimation of *lifetime expectancy* and the planning of maintenance intervals. Decisions on this levels are usually made by experts (Marwala, 2012).

## Overview of this thesis

*Roller bearing* datahas already been used in many publications, to evaluate and proof *condition monitoring* concepts (Marwala, 2012), (Nelvamondo, Marwala, & Mahola, 2006), (Li, Chow, Tipsuwan, & Hung, 2000). Most of these papers used both normal and outlier training data to evaluate the efficiency of *supervised* classifiers such as *neural networks* or *support sector* *machines*. Goal of this thesis is the design and evaluation of an exemplary *semi-supervised* *condition monitoring* approach, using the *roller bearing* data from (Case Western University) as benchmark.

My contribution involves

* Research on *condition monitoring* and a concise presentation of a general condition monitoring framework (1.1)
* Research into *feature extraction* methods and selection of suitable methods with respect to the benchmark data set (2,4.2)
* Implementation of a *Higuchi fractal dimension (HFD) feature* extraction method (4.2.2)
* Research on *one-class-classification* and selection of suitable classifiers in relation with the benchmark data set and the available features (3,4)
* Design and implementation of a new *one-class-classifier* based on *random forest* (3.7.6,4.3)
* Implementation and evaluation of a complete *semi-supervised condition monitoring* approach, including *feature extraction*, *feature selection*, classification and tests (4).

The first part of this thesis introduces several techniques, starting with *feature extraction* methods in chapter 2. Chapter 3 provides a general approach for the construction of *one-class-classifiers* and continues with an introduction of various *semi-supervised* classifiers which can be found in literature. The final part of this chapter discusses several aspects of the *random forest* classifier (3.7), of which a *semi-supervised* version was developed as part of this thesis (3.7.6), (4.3).

The techniques introduced in the first part, are evaluated in the second part of this thesis (4), which starts with a description of the *roller bearing* data set in 4.1. The *feature extraction* results are briefly discussed in 4.2 and 4.3 provides a proof of concept of the *semi-supervised random forest* version introduced in 3.7.6. Two experiments where designed to evaluate the complete *outlier detection* scenario, of which the first one compares the classifier performances (4.4.1), while the second analyzes the applicability of the *random forest variable importance* for feature reduction purposes.

# Feature Extraction

Vibration analysis is the most popular approach to m*achine condition monitoring* (Nelvamondo, Marwala, & Mahola, 2006). Consequently, many different methods have been devised to extract meaningful *features* from machine vibration data. Some of these methods, such as *peak level*, *crest factor* or *kurtosis*, measure statistical characteristics of the time signal. Other methods, such as the *Fourier* or the *wavelet transform*, transform the time signal into another domain to reveal data aspects hidden in the raw time signal.

In this thesis, a mixture of several techniques is used to extract features that can be used to distinguish normal from fault machine conditions. Two of these techniques, *kurtosis* and *multi fractal dimensions (MFD)* are based on the time signal directly. The third method, *mel frequency cepstral coefficients (MFCC)* includes a transformation of the time signal into the time-frequency domain.

## Mel Frequency Cepstral Coefficients

*Mel frequency cepstrum coefficients (MFCC)* capture the dynamic features of a signal by extracting both linear- and non-linear properties (Marwala, 2012). The technique was originally devised in the field of speech recognition, where the *mel*-scale is used to adapt the measured frequencies to the frequency response of the human auditory system.

The basic *cepstrum* is defined as the inverse *Fourier transform* of a logarithmic frequency spectrum:

(2.1.1)

Where is the inverse Fourier Transform and is the frequency spectrum of a time signal .

Insight into some useful properties of a *cepstrum* can be gained by regarding a time signal as the output of a *linear time invariant (LTI)* system, characterized by its *impulse response* . The output of an *LTI* is given by the convolution of the input signal and the impulse response:

(2.1.2)

In the Frequency domain, the convolution in (2.1.2) transforms into a multiplication:

(2.1.3)

With (2.1.1) and (2.1.3) the *cepstrum* of an *LTI* gives

The *cepstrum* of an *LTI* can thus be written as

(2.1.4)

From (2.1.4) it can be seen, that a *cepstrum* separates the input signal from the impulse response in the time domain.

Applied to the Condition Monitoring of machines, in (2.1.2) represents the signal measured by accelerometers at a certain point outside a machine. The actual information about machine conditions is mainly contained in the signal , which is generated somewhere inside the machine and filtered by its transmission path through the machine. The *cepstrum* isolates the condition signal in the time domain (Kolerus & Wassermann, 2008).

The *Mel Frequency Coefficients* used in various Condition Monitoring applications (Nelvamondo, Marwala, & Mahola, 2006), (Marwala, 2012) are based on a *mel*-transformation of a time-frequency spectrum and can be calculated by

1. Transforming the input signal from the time domain into the time-frequency domain through a windowed *Fourier transform*:

Where is the number of frames, , is the time signal in window n and is the *hamming window* given by

with and a normalization factor .

1. Changing the frequency spectrum to the Mel scale using the equation
2. Converting the logarithmic Mel Spectrum back to the time domain using the *discrete cosine transform*

where , is the number of frequencies extracted from the ith signal frame and is the transfer function of the nth filter on the filter bank (Marwala, 2012).

## Fractal Dimensions

*Fractals* are patterns consisting of sub-patterns which are equal or similar in shape to the complete pattern. A fractal shape is characterized by a *fractal dimension* which exceeds the topological dimension of the shape and may fall between two integer numbers (Mandelbrot, 2004).

Applied to the analysis of time series, *fractal dimension* can be seen as a measurement for the irregularity of a curve (Polychronaki, et al., 2010), which has a topological dimension of one and a *fractal dimension* between one and two.

Various algorithms exist to calculate the fractal dimension of time series, often based on some kind of distance measurements over several scales. Popular algorithms include the *box-counting* method, *Katz’s* *method* and *Higuchi’s* *method* (Raghavendra & Dutt, 2010).

*Fractal dimensions* have been successfully used as *features* in several research scenarios (Nelvamondo, Marwala, & Mahola, 2006), (Marwala, 2012). In this thesis, *Higuchi’s* method is used.

Given a discrete time series consisting of samples

the *Higuchi fractal dimension* *(HFD)* is calculated by (Esteller, Vachtsevanos, Echauz, & Litt, 2001)

1. Constructing new time series as

for

where indicates the initial time value, k indicates the discrete time interval between individual points and is the next lowest integer value to a.

1. Computing the average length for each of the new curve as

with and the normalization factor

1. Calculating the complete length for each k as sum of the average lengths:
2. Estimating the Fractal Dimension *D* as the slope of the least squares linear best fit line for the curve versus , since

## Kurtosis

*Kurtosis* is a general term for any measure of curve *peakedness*. In statistics, *kurtosis* is defined as the normalized fourth-order moment, given by

Where is the number of samples, is the mean and the variance.

Since machine failures often manifest themselves in vibration signal spikes, *kurtosis* can be a viable features extraction method in machine *condition monitoring* (Nelvamondo, Marwala, & Mahola, 2006).

# One-Class Classification

A *classifier* can be represented by a function

Where the data object to be classified is , is a parameter or weight vector and y is the estimated class for**.**

The parameters is usually inferred from a set of sample objects of the form (1.1), by optimizing some kind of training function. A typical approach is the minimization of an *error function* such as the *sum of squares error*, given by

where is the class label of the nth sample object .

In a supervised setting, where training samples are available for all possible classes, a classifier creates a *classification boundary* separating objects in *feature space*. Figure 2 illustrates a (non-optimal) *class boundary* for a two-class problem with the two *features* and .

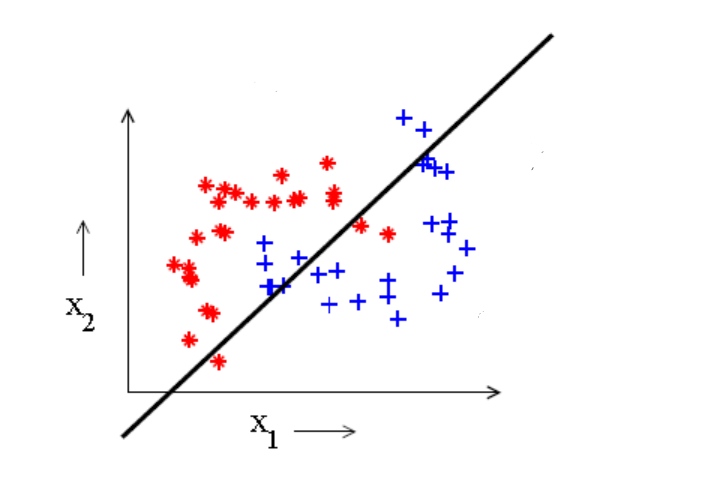


Fig. 2: Two-Class-Classification

The quality of a trained *classifier* is measured by its ability to correctly classify *unseen test data*, which was not used in the training of the classifier. This important property, also known as *generalization* (Bishop, 2009), depends on a number of factors. A fundamental aspect for (*supervised*) classification performance is the quality of the sample set used in training. If the samples do not represent the underlying population well, the classifier will perform poorly on unseen samples. Other important aspects are the choice of classifiers and the degree of adaption to the sample set during training, which is controlled by the number of parameters . The line illustrated in figure 2 is not flexible enough to capture a class distribution which is not linearly separable and thus always shows a *bias* towards a certain class distribution. If a classifier is too flexible on the other hand, the adaption to all the details in the training set (including the noise) may be too high, which is also known as *overfitting* and results in poor *generalization.* The problem of finding an optimal degree of complexity between these two extremes is referred to as *bias-variance trade-off* (Domingos, 2012).

If sample data is available for only one class, a different classification approach has to be chosen. Since nothing is known about the distribution of classes for which no sample data is available, the general goal in such a *semi-supervised* case is to find an optimal description for the known class, also referred to as *target class*. New objects can then be compared to this *data description* and accepted as *target class* members if they match the model closely enough, or rejected as outlier otherwise.

Figure 3 illustrates this concept with a closed *boundary* as *data description* for the “+”-class from figure 2.

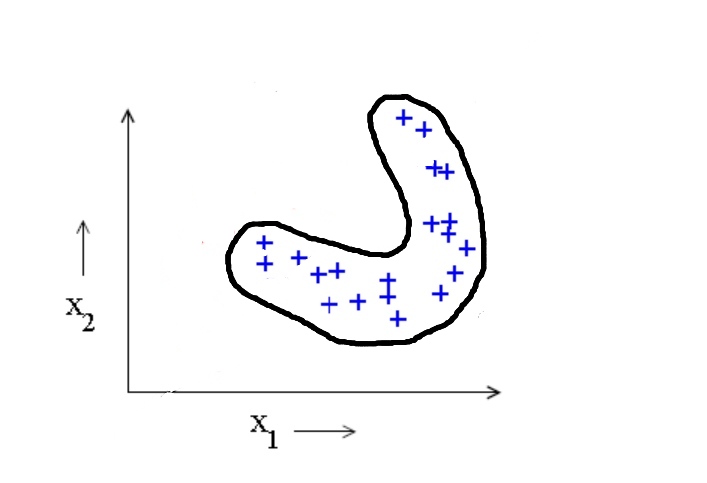


Fig. 3: One-Class-Classifier

This type of classifier can be represented by a mapping of the form

where label is applied to objects that belong to the target class and label to all others. Depending on the context, this classification approach is known as *one-class-classification*, *data description*, *novelty detection*, *outlier detection* or *concept learning*. A comprehensive introduction to the theory of *one-class-classification* can be found in (Tax M. J., 2001).

All *one-class-classifiers* applied in this thesis consist of two main components. The first component is a function measuring the relation of new data objects to the *target training data*. This function can be either some kind of distance measure or a *resemblance probability* . The second component is a threshold defining the decision boundary. New objects are accepted if their distance to the *target class* is smaller than the threshold

(3.1)

or if their probability of belonging to the target dataset is above the threshold

(3.2)

Where is the *indicator function*.

For a distance based method, the threshold can be defined by

(3.3)

where is the nth training sample and is the fraction of accepted target samples from the training data. For a probability based approach, the threshold is accordingly given by

(3.4)

The optimal value for has to be found experimentally. For some robustness against outliers in the training dataset, should be smaller than one. However, the smaller , the larger the classification error on the target objects.

For the evaluation of *one-class-classifier* performance, two types of errors have to be considered. *error type 1* is defined as

where is the fraction of rejected targets, and *error type 2* is defined as

Where is the number of accepted outliers.

The distance measure and the probability measure were introduced above as general measures for the similarity of objects to a known *target* class. The algorithms by which these measures are calculated have to be defined by each *one-class-classifier* individually. Some common *one-class-classifiers* are briefly described in the following chapters. Chapter 3.7 introduces a concept for a new *one-class-classifier* based on *random forest.*

## Support Vector Data Description

For linearly separable 2-class problem in a 2-dimentionalfeature space, a *support vector machine* finds an optimal boundary line by maximizing the perpendicular distance (or *margin*) of the points closest to the line. This principle is illustrated in figure 4.



Fig. 4: Support Vector Machine

The *support vectors*, marked by circles in figure 4 define the optimal *decision boundary* (in this case a simple line) from both sides. Generalized two *D* dimensions, the *decision boundary* is a hyperplane of the form

(3.1.1)

The optimal values for the parameters and can be found by solving (Bishop, 2009)

(3.1.2)

Where is the class label of the nth training sample .

After a series of optimization steps, the final *support vector* classifier is given by

(3.1.3)

where the training vectors that correspond to an are support vectors and is a *kernel* function (Bishop, 2009). The class membership of an unknown test point is indicated by the sign of (3.1.3).

According to the general definition of *one-class-classifiers* given above, the *semi-supervised* version of a *support vector machine* - also known as *support vector data description (SVDD)* (Tax & Duin, 2004) – tries to find a boundary enclosing all or most of the *target class* training samples. This optimal *boundary* is a hypersphere with radius R and center **a** as illustrated in figure 5.

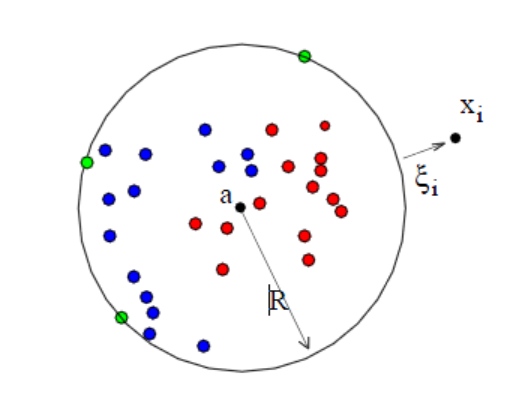


Fig. 5: Support Vector Data Description

The goal in *SVDD* is the minimization of the radius such that the hyperspherecontains all training data points, which is expressed by the constraints

(3.1.4)

Where and are the center and the radius of the hyperspere and is the n-th training data vector.

For robustness against outliers in the training data set, the constraints in (3.1.4) have to be relaxed by so-called *slack-variables* , which extends (3.1.4) to

(3.1.5)

The effect of slack variables is indicated in figure 5 by an outlier and the corresponding slack variable .

After a sequence of optimization steps (Tax M. J., 2001), the final *SVDD* classifier is given by

(3.1.6)

where is the test object to be classified, and are the nth and the mth training vectors and and are Lagrange multipliers which are only nonzero, if the corresponding and are support vectors.

The vector products in (3.1.6) can be replaced by *kernels*, resulting in the classifier

A common choice for the *kernel* function is the *Gaussian kernel*, defined as

Figures 6 shows an *SV-dd* boundary calculated with a *Gaussian kernel* for a banana shaped target data set with 2 features.

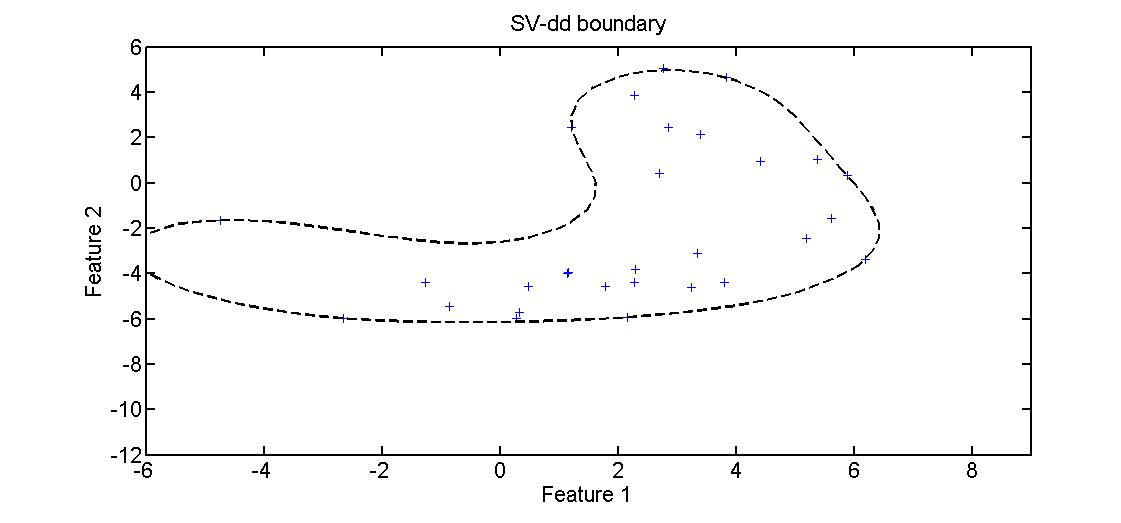


Fig. 6 SVDD with Gausian kernel

## K-Center

*K-Centers* is based on the *domain approximation* algorithm (Ypma & Duin, 1998), where a subset of *k* support objects is selected from the *N* samples of a training dataset ***X*** and each of the support objects is surrounded by a *receptive field* of equal radius *r*. A sample object of the dataset ***X***lies in the *receptive field* of a support object if

and

I.e., each training object lies in the receptive field of the nearest support object, as long as the distance to that support object is smaller than the radius .

The problem of finding the subset with the minimal radius *r* such that all data samples in are covered by the receptive field of one support object is tantamount to minimizing the error

The optimization algorithm starts with randomly chosen support objects and uses a search forward strategy. To avoid suboptimal solutions, the process is repeated several times with different random initializations and the solution with the lowest error is used (Tax M. J., 2001). The parameter and the maximal number of retries have to be supplied the user.

With the support objects determined in the training phase, the distance of a *test object* to the target set can be calculated by

This distance measure can be used with the classification approach defined by (3.1) to classify new objects.

Figures 7 shows an *K-Center-dd* boundary calculated for a banana shaped target data set with 2 features.

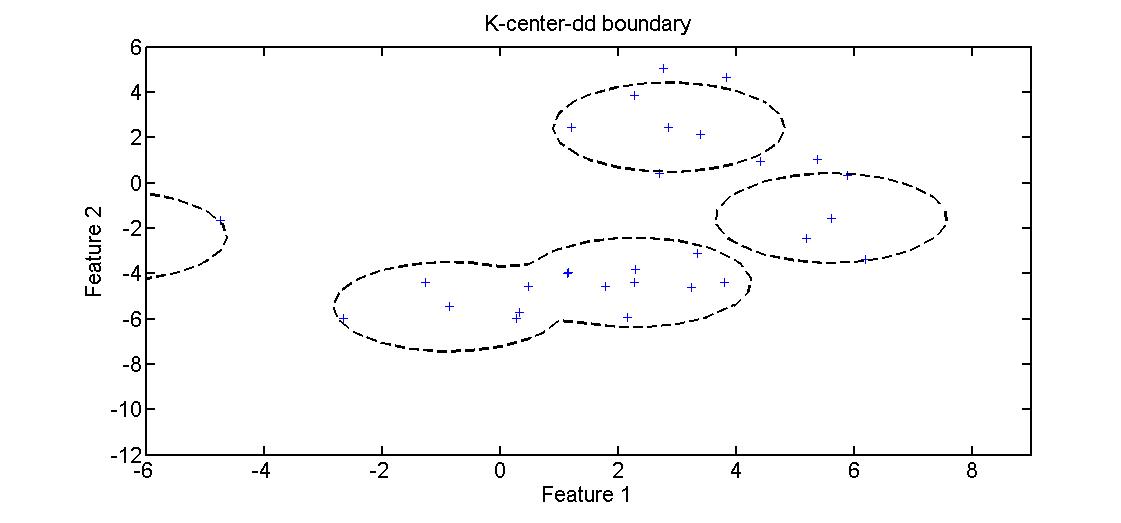


Fig. 7 K-Center boundary

## K-Means Data Description

*K-Means* is a clustering algorithm similar to *k-centers* where *k prototypes* are defined and the clustering is based on a distance measure between training data objects and the nearest prototypes. The main differences to *k-centers* are the placement of the *prototypes* - which in *k-Means* is not constrained to coincide with existing training data objects - and the *error function*, which is defined by

where is the nth of training objects.

The minimization of implies a placement of the *k* prototypes such that the distance of each training object to the closest *prototype* is a minimum. A typical algorithm for this optimization problem starts with a random placement of *k* *prototypes* and calculates *k* clusters, consisting of the training data objects closest to the respective *prototype* . Then for each cluster, a new *prototype* is calculated by simply taking the mean of the data objects in that cluster. This procedure is reiterated until the *prototypes* are stable.

Just as in *K-Centers*,the relation of new objects to the target data is determined by their squared distances to the nearest *prototype*

Figures 8 shows an *K-Means-dd* boundary for a banana shaped target data set with 2 features.

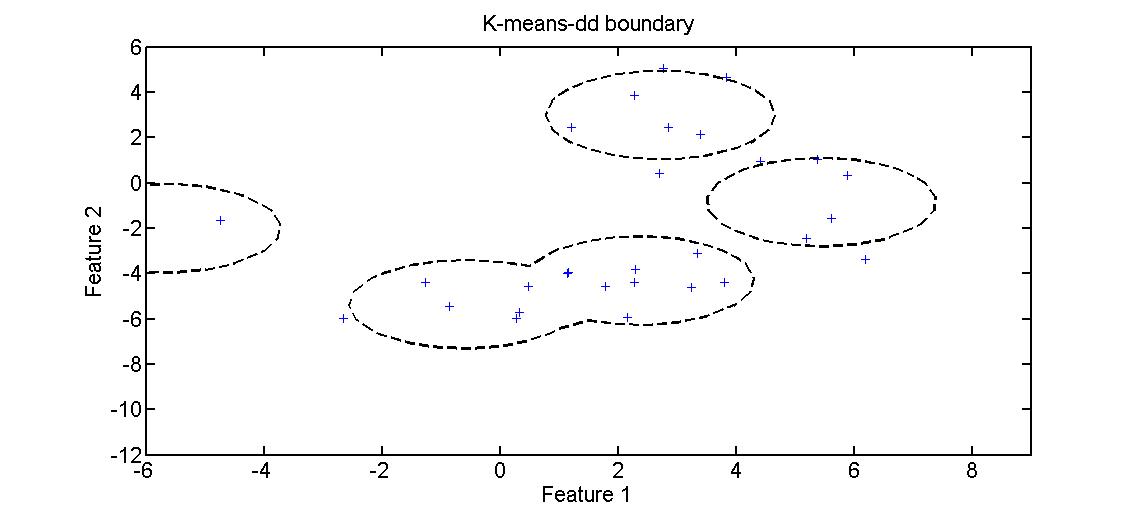


Fig. 8 K-Means-dd boundary

## Nearest Neighbor Data Description

*K-Nearest-Neighbor* classification is based on a local density estimation of the form (Bishop, 2009)

Where K is some fixed integer value, *N* is the size of the training dataset and can be interpreted as the volume of a small sphere centered on a data object **.** With the availability of supervised samples, a test value can be classified by inflating until the sphere contains exactly samples and assigning the majority class of all data points in the sphere to .

In an adaption of the *K-Nearest-Neighbor* method to *semi-supervised* problems (Tax M. J., 2001), the distance of a test object to its nearest neighbor in the sample set is compared to the distance between the nearest neighbor of and the nearest neighbor of the nearest neighbor of . The test object is accepted if the distance is smaller than or equal to the distance .

This can be formalized by the *classifier* function

Figures 9 shows an *NN-dd* boundary calculated with a *Gaussian kernel* for a banana shaped target data set with 2 features.

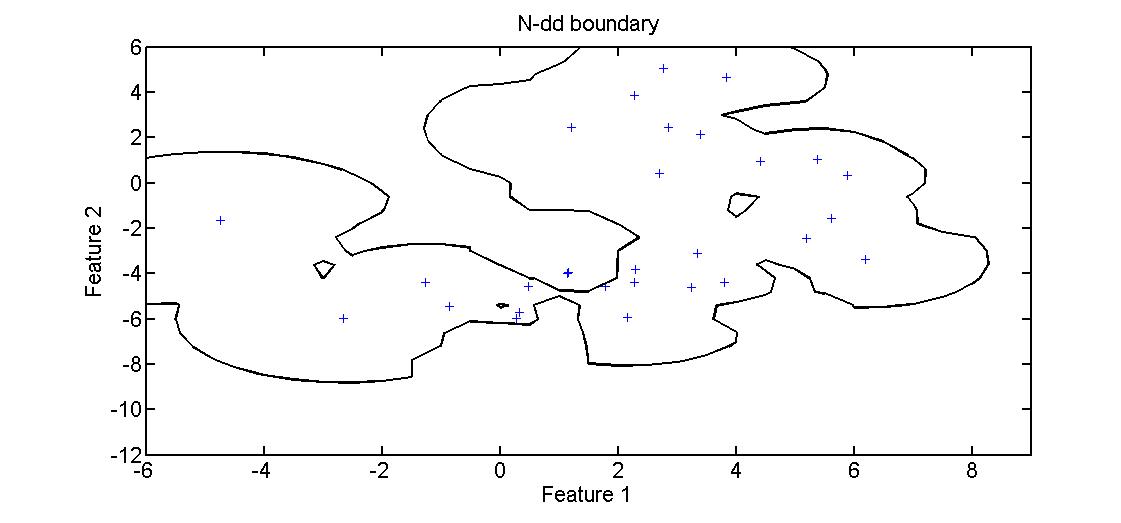


Fig. 9 NN-dd boundary

## Parzen Window Data Description

In its general form, a *Parzen Window* estimator is given by (Bishop, 2009)

(5.7.1)

where is the nth training point, and is a *kernel function* with the width , which has to satisfy the constraints

and

With a *Gaussian kernel*, the *Parzen Windows* estimator is given by

Where is the width, which can be optimized by a maximum likelihood method (Tax M. J., 2001). The *Parzen window* density estimation (5.7.1) can be used with the classification approach defined by (3.2) and (3.4) to classify new objects.

Figures 10 shows a *Parzen-dd* boundary calculated for a banana shaped target data set with 2 features.

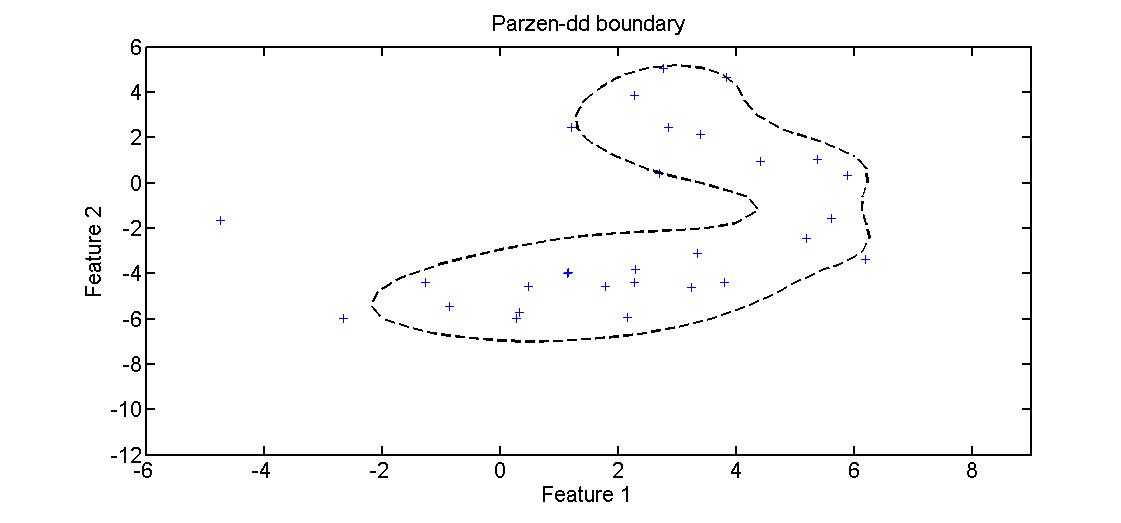


Fig. 10 Parzen-dd boundary

## Self Organizing Maps

A *Self Organizing Map* consists of several nodes with a defined position in an *N-*dimensional grid structure. Each node in the grid is associated with a *D-dimensional* *prototype* representing *D-dimensional* sample data objects.

*N* is often chosen to be 1 or 2, such that

The positions of the nodes in the *N-dimensional* grid are supposed to reflect the neighborhood relations of their associated *prototypes*, such that neighboring nodes in the grid are eventually associated with neighboring prototypes in data object space. Through this topology preserving property, *SOM*s create a mapping from a high dimensional *input space* to a low dimensional *feature space*, and allow indirect visualization of relations in a high dimensional data space (van der Heijden, Duin, de Ridder, & Tax, 2004).

The *SOM* training starts with an initialization of the prototypes where denotes the node number and is the *prototype* associated with node . Initialization can simply be done by randomly assigning a training data object as *prototype* to each node.

Then, for each sample data object **,** the following two steps are executed:

1. Detection of the node whose associated *prototype* is closest to by calculating

Where n denotes the sample data object and the iteration counter.

1. Updating of the winning node and its grid neighbors by computing for each node *j*

Where is a learning rate which can be set to decline with the iteration count and is a weighting function which determines by how much a node *prototype* is updated. The input to this weighting function is the distance between the winning node determined in step one and the node whose *prototype* is to be updated. The weighting function has to satisfy the constraints

and

to ensure that the winning node receives the largest update.

A common choice for the weighting function is the Gaussian function

Where defines the width of the neighborhood area, which can decrease with each iteration step.

Other choices that have to be made prior to the training are the structure and dimension of the grid and the number of nodes. Often, a 2-dimensional grid is chosen for visualization purposes. In many cases the best combination of parameters has to be found experimentally (van der Heijden, Duin, de Ridder, & Tax, 2004).

To classify a test object, its distance to the closest prototype is measured. If this distance is below the predefined threshold, the test object is accepted as member of the target class and rejected otherwise.

Figures 11 shows an *SOM-dd*  boundary calculated for a banana shaped target data set with 2 features.

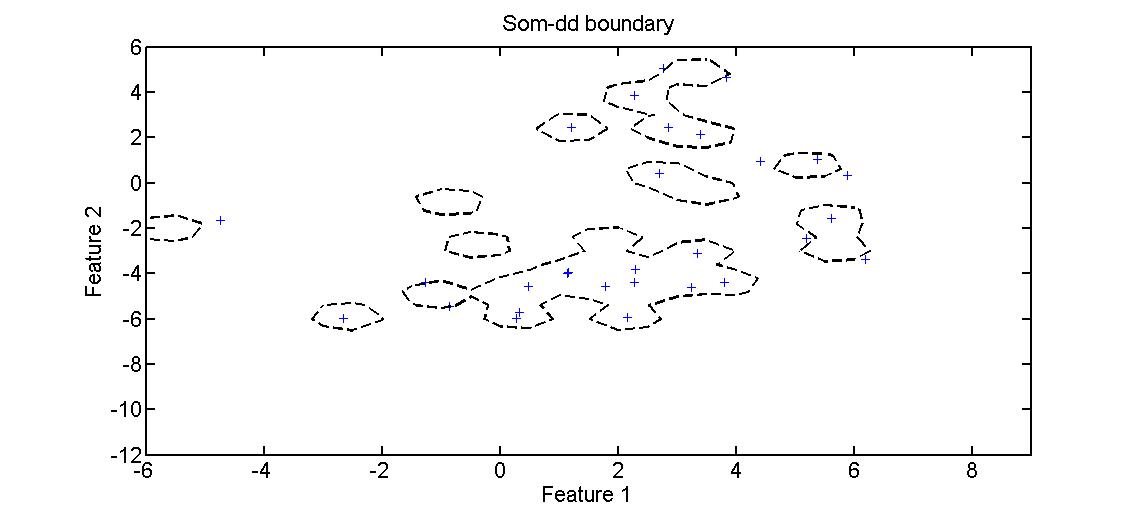


Fig. 11 SOM-dd boundary

## Random forest

*Random forest* (Breiman , Random Forests, 2001)is an ensemble learning method for classification and regression, which grows many trees from sampled subsets of a data set. *Test objects* are run down all trees in the forest and labeled by a majority vote.

Each tree in the forest is grown in the following way:

1. If the number of cases in the original data set is *N*, *N* cases are sampled from the training set, with replacement.
2. If the number of features in the data set is *M*, a number is chosen. For each node in the tree attributes are randomly sampled from the original M attributes and the best split among these m attributes is used to calculate the split for the node.
3. Each tree is grown to maximum size, without pruning.

The individual trees are grown using the *CART* algorithm, with an additional randomization moment in the attribute selection at each node, as described in step 2 above. *CART* is a decisions tree algorithm which uses the *gini criterion* to split nodes and only allows binary splits (Quinlan & Kohavi , 1999). The split decision is made by selecting at each node the attribute resulting in the highest *information gain*

(3.7.1)

Where S is the dataset before splitting, are the splitted datasets and is the *gini index*

(3.7.2)

where is the relative frequency of cases with class in dataset .

*Random forests* have a number of useful features, making them a good choice for a variety of classification and regression problems. Among the advantages of random forests are

* A higher *accuracy* than many other algorithms
* The abiltiy to run on large datasets
* The ability to process thousands of attributes without the need of feature selection
* An implicitely calculabe *generalization error* without the need of methods such as *cross validation*
* The capability to calculate a *proximity* between data objects, which can be used to cluster data or to detect outliers
* Efficient methods for replacement of missing data
* The ability to measure attribute importance, which can help to reduce the datasets for other applications
* A posibility to find class prototypes based on the *proximity* measure, which can help to gain more insight into the data

Some of the features are a result of the basic tree growing process, while others are based on optional computations during tree construction.

For the construction of each tree in a random forest, a set of N samples is uniformly sampled from the original data set with replacement. Thus, a certain number of sample data is left out in the construction of each tree. This *out of bag (oob)* data (Breiman , Random Forests, 2001) can be used for a simple estimation of the generalization error in the following way:

1. For each tree all sample objects which were not used in its construction are put down that tree to get a classification.
2. Each sample object is assigned the class label that got the most votes during the test run in step 1.
3. The generalization error is estimated by dividing the number of samples where the labels applied in step 2 did not match the real classes by the complete number of samples.

### Variable importance

A measure for the *variable importance* can be obtained by randomly permuting the values of the *features* in the dataset as follows:

1. Each training data object is put down the trees where it was *oob* (i.e. left out in the training) and the overall number of correct votes is counted by variable *c*.
2. The values of one *feature* *m* in the dataset are randomly permuted.
3. The first step is repeated with the permuted dataset and the number of correct votes is counted by *p*.
4. The importance of feature *m* is estimated by

where is the number of trees in the forest.

### Proximity

One of the most useful features in *random forests* is the *proximity matrix* which is an matrix, where is the number of data objects used in the forest construction and each entry is a measure for the similarity between objects and . The matrix entries are computed during the forest construction in the following way:

1. After a tree is build, the complete *training* *set* is put down that tree and classified.
2. Whenever a tree assigns the same class to training objects and in step 1, the value is incremented by 1.
3. After the complete forest is constructed, the entries are normalized by the number of trees in the forest.

Several other useful featuresare based on the proximity matrix, such as the *prototypes* or the outlier measure.

### Class prototypes

*Prototypes* can be calculated based on the *proximity matrix* by

1. Identifying the training data object with the most other class *c* training data objects among its *k* nearest neighbor
2. Taking the median values for each feature from the *k* objects identified in step 1 and constructing a prototype object with these medians.

### Outlier measure

*Random Forest* defines outliers with respect to their class membership as training data objects, whose proximity to all other objects in the same class is small.

The average proximity of a an object in class to all a other objects in class is

Based on this average proximity, a raw outlier measure for point is defined as

Where is the number of samples in the training set.

The final *Outlier Measure* of a point is calculated by

Where is the median of all raw outlier measures of objects in class and is the *Median Absolute Deviation* of all raw outlier measures in class .

### Unsupervised training

In case of unlabeled training data, the random forest method considers all training data to belong to class 1 and creates a synthetic second dataset of the same size and dimension as the training data set. All samples of the synthetic dataset are labeled as class 2. A synthetic data object of class 2 is created by uniformly sampling from the value distribution of each feature of the original training data set.

The original training data and the synthetic data can then be combined and used to construct a two-class random forest including most of the standard options such as the *proximity Matrix,* *outlier detection* and *variable importance*.

### Semi-supervised classification

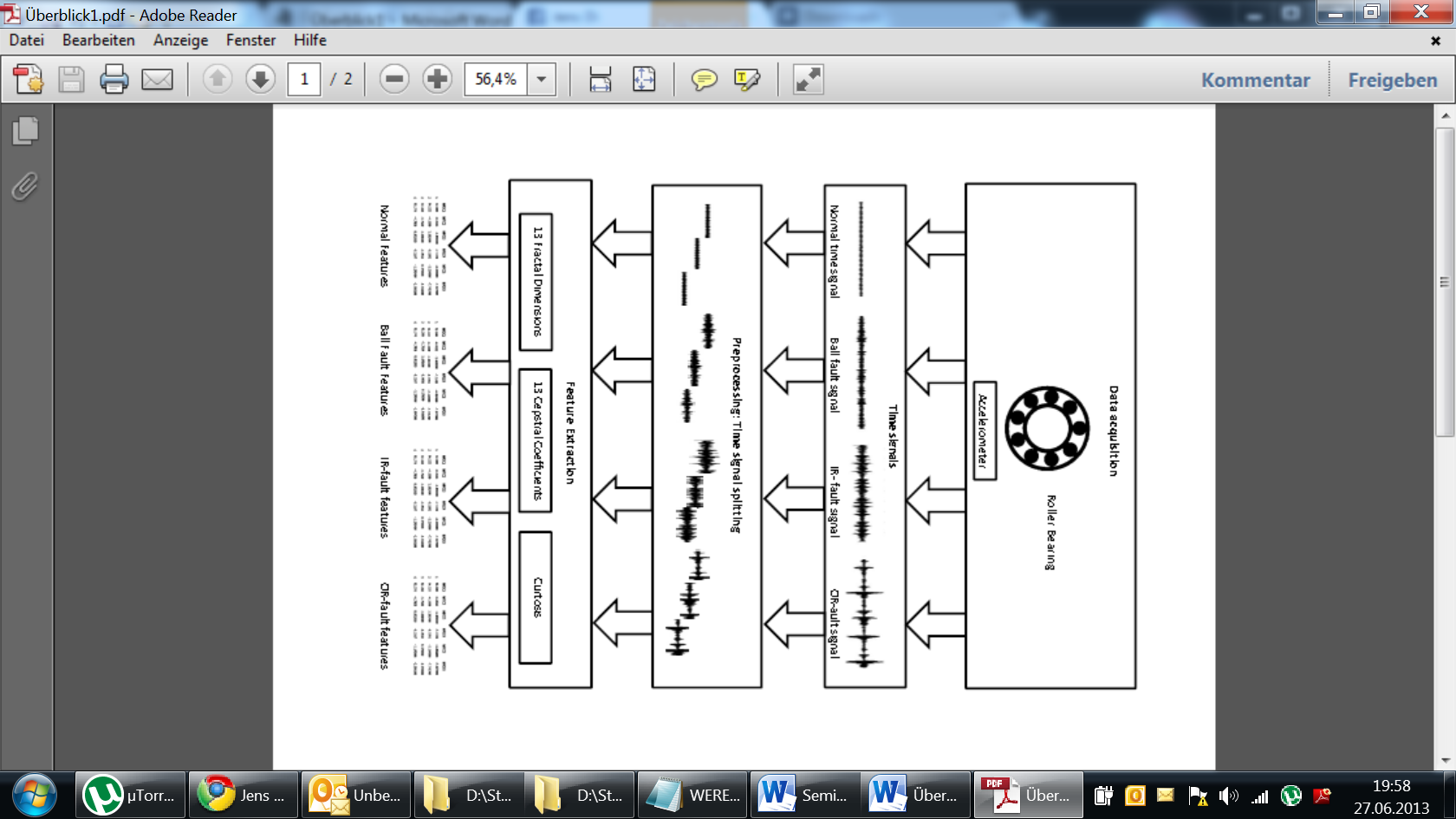
A basic approach for *random forest* based *one-class-classification* can be outlined as follows:

* Construction of a random forest in unsupervised mode using the target training data
* Calculation of one or more class prototypes based on the proximity matrix
* Calculation of the distance between new test objects and the prototype(s) according to the approach introduced in chapter 3
* Classification of test objects as targets if their distance to the closest prototype is below a certain threshold and as outliers otherwise (as defined by (3.1))

An evaluation of this concept can be found in 4.3.

# Experiments

Figure 12 provides an overview of the steps involved in the classification and evaluation approach of this thesis.



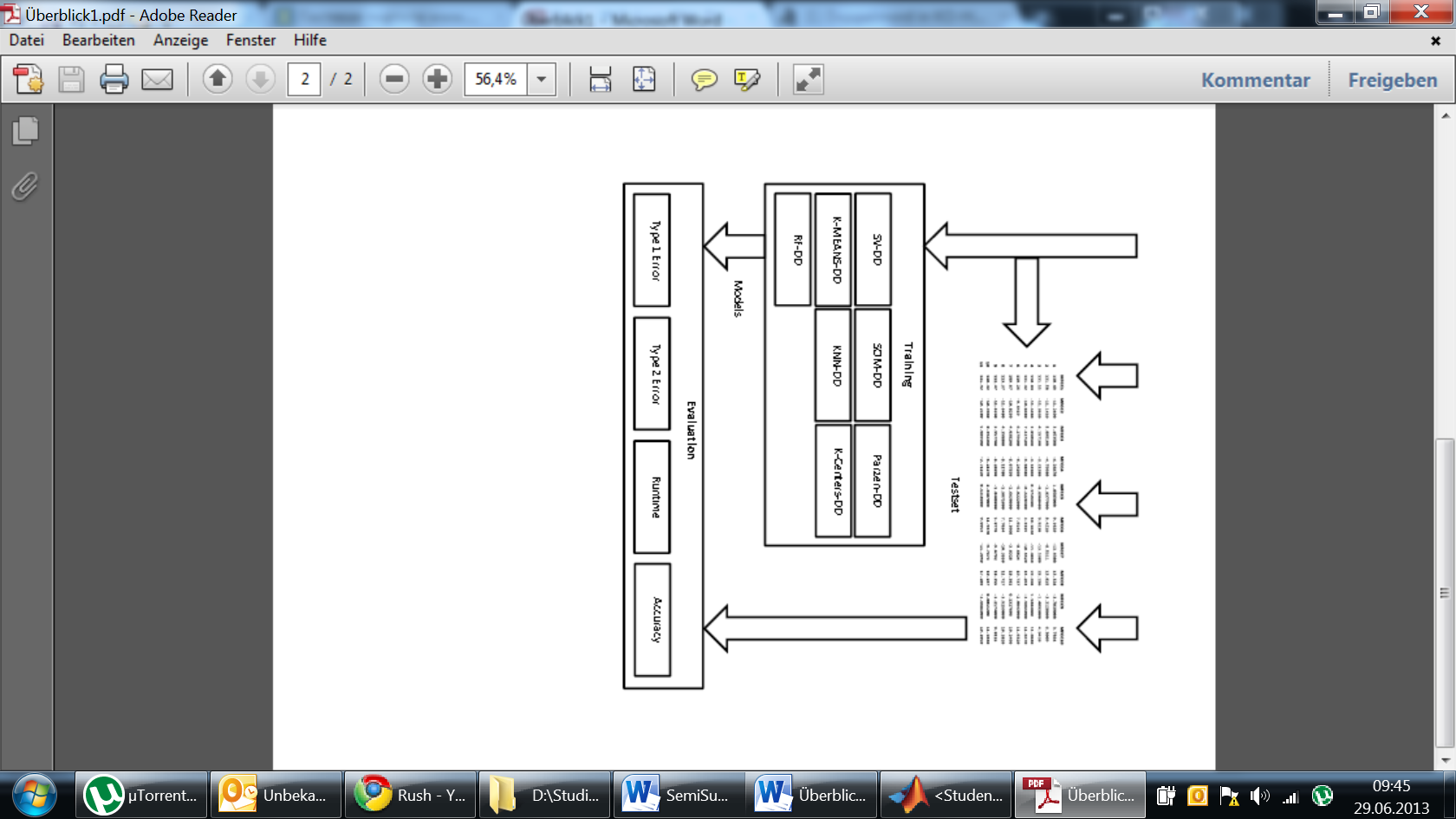


Fig. 12 Experimental Sequence

Data Acquisition was not part of this thesis, the roller bearing data sets available at (Case Western University) were used as benchmark. A brief overview of these datasets is given in 5.1. The preprocessing and feature extraction steps are summarized in 5.2 and chapter 5.3 describes the classification and evaluation sequences.

All experiments were completely implemented as MATLAB scripts. Except for the *Random Forest* classification, all *One-Class-Classifiers* were realized with *mappings* from the *Dd\_tools* toolboxprovided by the TU Delft (Tax D. M., 2012). The *Random Forest* approach as outlined in 5.3 was first evaluated based on an R-package (Liaw & Wiener, 2002) and then integrated into the *Dd\_tools* framework.

## Roller Bearing Datasets

Rotating Machines are very common in various industrial applications. In manufacturing, most machine failures are linked to bearing faults (Lou, Loparo, Discenzo, Yoo, & Twarowski, 2004). Consequently, much effort has gone into the development of classification and prediction techniques for bearing faults (Marwala, 2012), (Nelvamondo, Marwala, & Mahola, 2006), (Li, Chow, Tipsuwan, & Hung, 2000).

Roller bearings consist of two concentric rings, the inner and outer raceway (*IR* and *OR*), with a set of rolling elements running between their tracks, as illustrated in Figure 13.

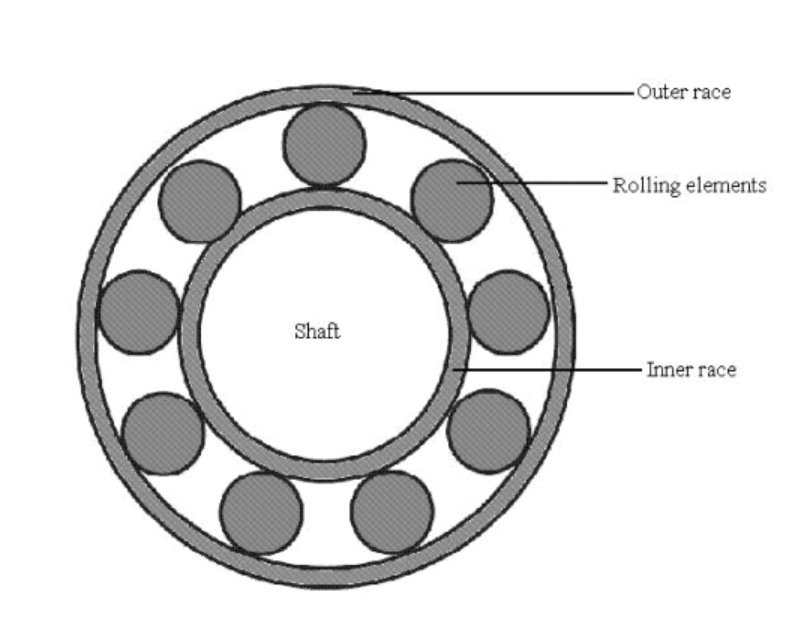


Fig. 13 Roller Bearing

The rolling elements are usually contained in a cage to prevent contact between elements.

*Roller bearings* generate vibration signals with characteristic shapes, depending on the conditions of the raceways and the rolling elements. Faults in one of these parts typically manifest themselves in characteristic frequencies in the vibration signal (Li, Chow, Tipsuwan, & Hung, 2000).

The *roller bearing* data set collection provided by (Case Western University), contains vibration signals representing the following *roller bearing* conditions:

* Normal conditions
* Ball fault conditions
* Inner raceway fault conditions
* Outer raceway fault conditions

The individual datasets in the collection are defined by the parameters

* Position of the accelerometer for data acquisition (fan end or drive end)
* Rotation speed (1797rpm,1772rpm,1750rpm,1730rpm)
* Motor Load, correlated to the Rotation speed (0HP,1HP,2HP,3HP)
* Sample rate (12k and 48k)

In this thesis, for each condition the dataset acquired from the drive end, at a speed of 1797 and with a sample rate of 48k was chosen. Figure 14 displays sequences of each data set corresponding to the first five revolutions of the roller bearing.

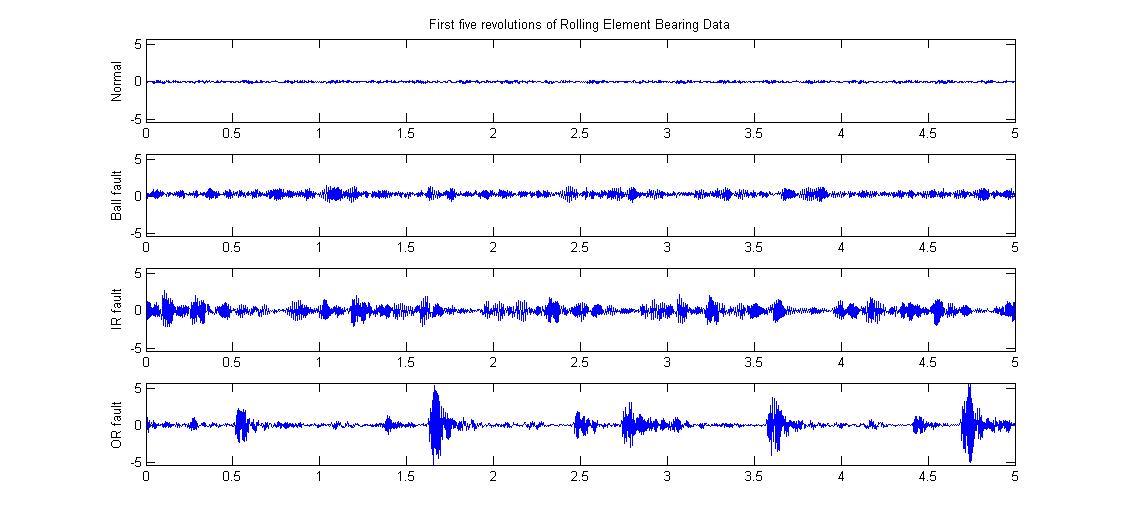


Fig. 14 Roller Bearing Sample Data

To simulate a *semi-supervised* scenario, only the normal condition data was used in classifier training*.* The three fault data sets were used in the evaluation of the trained models exclusively.

## Feature Extraction

In a preprocessing step, the large data sets representing the normal and the three fault conditions were split into equal segments with a length corresponding to five roller bearing revolutions. The Kurtosis was calculated directly from each segment. For the extraction of MFD and MFCC, each segment was further split into 15 frames of equal length. Of each of these frames, 13 *MFCCs* and 13 *HFDs* were extracted.

Table 1 illustrates the schema of each of a feature sets.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **…** |  |  | **…** |  |  |
| **Feature Vector 1** | ### | ### | ### | ### | ### | ### | ### |
| **…** | ### | ### | ### | ### | ### | ### | ### |
| **Feature Vector N** | ### | ### | ### | ### | ### | ### | ### |

Table 1 Feature data set schema

### Mel Frequency Cepstral Coefficients

The MATLAB package used for extracting the *MFCCs* (Ellis, 2005)is part of a collection designed for feature extraction in the speech recognition domain. It is highly adaptable through a large number of parameters of which many require specific domain knowledge. For the purposes of this thesis, default settings proved to be sufficient for most of these parameters. The number of *Short Time Foruier Transform* frames was set to 15 and the number of *MFCCs* extracted from each of these frames was set to 13.

Figure 15 shows 13 *MFCCs* from the same frame of each dataset.

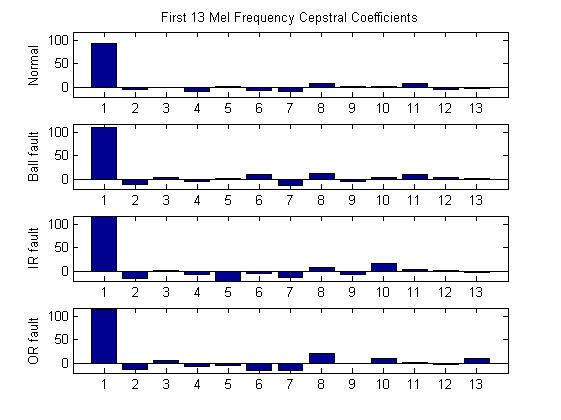


Fig. 15 Thirteen Mel Frequency Cepstral Coefficients

### Higuchi Fractal Dimensions

The *Higuchi Fractal Dimension* method was implemented according to the definition in (Polychronaki, et al., 2010) . A Weierstrass function with known *fractal dimension* was used to verify correct behavior of the *HFD-*function.

Since no exact rules exist for the selection of the free parameter , the parameter was varied within a certain range, based on experimental results (Polychronaki, et al., 2010). In order to get the same number of *HFDs* as *MFCCs*, each segment was split into 15 frames and for each frame 13 different *HFDs* where calculated by selecting a different value for each time. With this approach, the best values could later be selected by *PCA* or similar *feature selection* methods.

The graph in figure 16 illustrates *HFD* values of the first 15 feature vectors of each set, with .

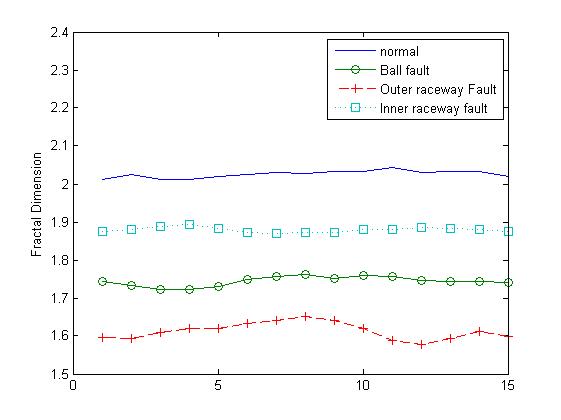


Fig. 16: 15 Higuchi Fractal Dimensions with k=6

The *fractal dimensions* of the normal signal obviously exceed the expected fractal dimensions of a curve, which should range between one and two. The reason for this may be related to the specific shape of the normal signal, which looks like random noise and shows no discernible patterns. However, what really matters in the context of feature extraction are not absolute feature values, but values which are well suited to distinguish conditions. Figure 16 suggests that *HFDs* are a good choice in terms of feature extraction for the given data sets.

## Random Forest Data Description

The *Outlier Measure* of *Random Forest* as introduced in 4.6 is only defined for the training data set and cannot be applied to unseen test data. It can however be used to identify critical or implausible data in the training set. Such data can then be removed or modified before a *Random Forest* retraining run.

Fig. 17 shows a plot of the *Outlier Measures* for a *Random Forest* trained with 427 roller bearing normal feature vectors.



Fig. 17 Outlier Measures of Roller Bearing normal training data

According to a rule of thumb given in (Breiman & Cutler, Random Forests), data points with an *Outlier Measure* beyond a threshold of about require closer inspection. Figure 14 shows clearly, that the *Outlier Measures* for all training data points are smaller than. Consequently, no further processing of the training data set was required.

Another useful feature which provides some insight into certain aspects of training data, is the *variable importance* measureintroduced in 4.2. Figure 18 shows the *variable importance* measures calculated during a *random forest* training with 427 normal feature samples.



Fig. 18 Attribute Importance of Roller Bearing normal features training set

The *attribute importance* can be defined as mean decrease in *accuracy* or as mean *gini* decrease, with both measures resulting in a different importance order of the attributes. An application of the *importance* measures as feature reduction method was evaluated as part of this thesis (5.5.2).

For the application of *random forests* in the *semi-supervised* *outlier detection* scenario assumed in this thesis, a *random forest* *one-class classifier* was implemented according to the generic approach introduced in 3. The classification sequence involves the following steps:

1. Training of a *random forest* in unsupervised mode (4.5), with a subset of the normal feature set
2. Calculation of a normal prototype (4.4)
3. Computation of the Euclidean distances between test objects and the normal prototype
4. Calculation of a distance based threshold according to definition (3.3)
5. Classification of the test objects as defined by (3.1)

Figure 19 illustrates the Euclidean distances between the normal prototype and a test set consisting of 120 samples, as calculated during a test run of the classification approach described above.



Fig. 19 Euclidean Distances between test data and normal prototype

The plot of Euclidean distances shows, that a constant classification threshold simply defined as the highest Euclidean distance among all distances between normal samples and the normal prototype (i.e. ), was perfectly suited to separate normal and outlier roller bearing data. For robustness against outliers in the training set, the threshold should be calculated with .

## Classification and Evaluation

All experiments were conducted on an Intel(R) Core(TM)2 Duo CPU P7450 machine with a clock speed of 2.13GHz and 8129 MB of RAM.

Two experiments were defined to evaluate the semi-supervised outlier detection approach introduced in this thesis. In a first experiment, the classifiers were trained and tested repeatedly to estimate several performance measures. Goal of a second experiment was to evaluate the applicability of the *random forest* *variable importance* measure for feature reduction. The two experiments and the results are described below.

### Performance evaluation

One experimental sequence comprised the following steps:

* Construction of a training set with 368 objects, sampled without replacement from the normal features
* Construction of a balanced test set with 180 objects, by combining the normal objects not used for training with 30 objects sampled without replacement from each of the fault feature sets
* Training of each classifier with the training set
* Testing of each classifier using the test set
* Taking the training and testing runtime as well as the error type I (fraction of rejected normals) and error type II (fraction of accepted outliers) for each classifier

The complete sequence was run ten times.

To compare classification results, several performance measures were calculated for each classifier (averaged over the ten runs):

* training and testing time
* type I and type II errors

Table 4 contains a summary of the results of one complete experimental run, where was set to 0 for all classifiers.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Parameter**  **Settings** | **Average Training**  **Runtime** | **Average Testing**  **Runtime** | **Fraction of Rejected Normals (e 1)** | **Fraction of Accepted Outliers (e 2)** | **Accuracy** |
| **RF-dd** | mtry=5 | 0.9432 sec | 0.0057 sec | 0.1044 | 0 | 0.9478 |
| **SV-dd** | Gaussian kernel | 20,2460 sec | 0.0093 sec | 0.5300 | 0 | 0.735 |
| **K-means** |  | 0,019 sec | 0.005 sec | 0.1422 | 0 | 0.9289 |
| **K-center** | #tries=25  K=5 | 0,4289 sec | 0.446 sec | 0.1100 | 0 | 0.945 |
| **N-dd** | - | 0,4585 sec | 0.013 sec | 0.1278 | 0 | 0.9361 |
| **Parzen** |  | 0,2277 sec | 0.0259 sec | 1 | 0 | 0.5 |
| **Som** |  | 162,045 sec | 0.0058 sec | 0.1733 | 0 | 0.9133 |

Table 4 Experimental results

Figures 20 and 21 compare the time performances and accuracies of the classifiers.

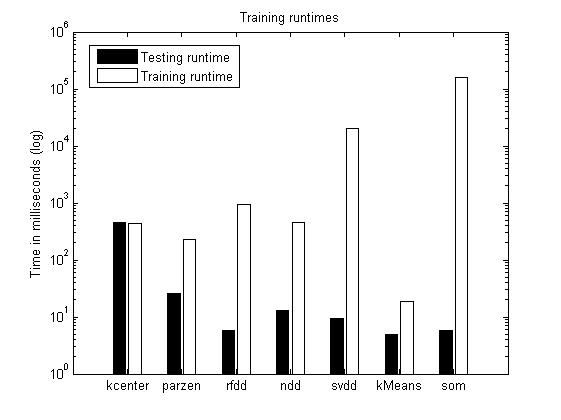


Fig. 20 Time Performance

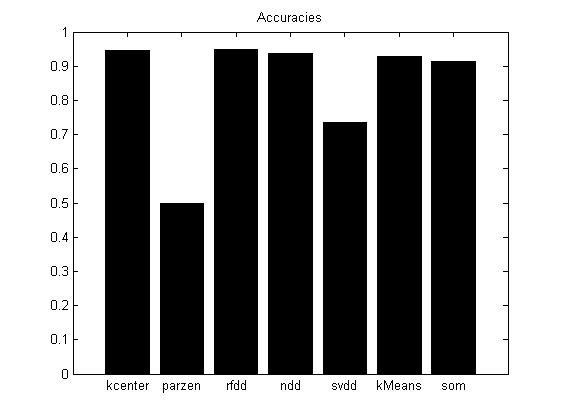


Fig. 21 Accuracies

A log-scale is used for the illustration of time performance values, since the training and testing times of the individual classifiers varied on a wide scale. The *SOM-dd* had by far the longest average training time, followed by *SVDD*. Fastest in terms of average training time was *K-Means-dd*. In a typical real world application, where classifiers would be trained offline and only once, the time needed to classify a new object is probably the more important measure. Best in this category was again the *K-Means-dd*, *K-center*, whose training and testing time were approximately equal had by far the longest testing time.

Five of the seven classifiers tested achieved an accuracy score above 0.9, *rf-dd* had the best results with an accuracy of 0.9478. The two worst classifiers in terms of *accuracy* were *SVDD* and *Parzen-dd*. *Parzen-dd* rejected all objects of the balanced test set and thus achieved an accuracy of only 0.5. A feature scaling prior to training of the *Parzen-dd* did not improve performance significantly.

### Feature reduction experiment

The feature reduction experiment involved the following steps:

* Construction of a training and a test set as in the performance experiment
* Training of a random forest classifier, calculation of variable importance
* In a loop starting with the most important feature, adding the next most important feature in each iteration until the feature set is complete:
  + Training of each classifier (except random forest) using the reduced training set
  + Testing of each classifier using the reduced test set
  + Calculation of error type 1 and error type 2 for each classifier

This complete sequence was repeated several times, with a different sample training and test set at each run. The resulting averaged accuracies for each reduced feature set are illustrated in figure 22.

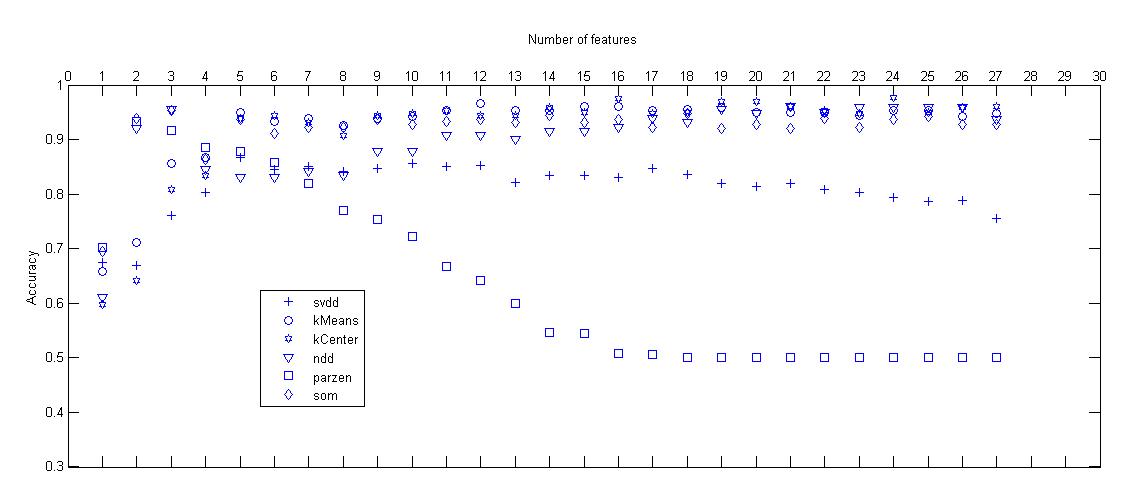


Fig. 22 Accuracies with reduced features

Figure 19 shows, that for most classifiers the number of features could be reduced considerably, without sacrificing accuracy. This could be an indication for redundancies in the feature set, which in turn would be the result of correlations between individual features.

The *Parzen* window showed an opposite trend compared with most other classifiers. Starting from a feature number of 16, its accuracy improved gradually until a maximum is reached at a feature number of 3. This could be a interpreted as a manifestation of the *curse of dimensionality*, which basically states that with a fixed number of training samples, the predictive power of a classifier decreases, as the feature number increases. To a lesser degree, this could also explain the SVDD curve, which shows a slight increase of accuracy up to a number of 5 features.

The outcome of this experiment proofs, that the *random forest* *variable importance* could be used as a suitable feature reduction methods in this *semi-supervised* setting.

# Conclusions

This thesis presented a completecomputerized *semi-supervised condition monitoring* scenario, including data preprocessing, *feature* extraction, *feature* reduction, *one-class-classification* and evaluation. The combination of *feature* extraction and classification methods provided altogether good results for the chosen roller bearing benchmark set.

The *random forest* based *one-class-classification* approach introduced in this thesis proved to be highly successful applied to the roller bearing data set. Additionally it was shown, how some of the *random forest* features could be used to gain some insight into data set characteristics in a *semi-supervised* setting. The general performance of the *rf-dd* classifier can be improved by calculating more than one *target class prototype* which would extend its applicability to more complicated *one-class-problems*. Research on how these prototypes can be calculated based on *random forest* features could be part of a future thesis on this subject.

Another interesting subject based on the results of this thesis could be the combination of *one-class-classifiers* in *batch-* or *boosting* methods.

# References

Banerjee, A., Chandola, V., & Kumar, V. (2007). Outlier Detection: A Survey.

Bishop, C. M. (2009). *Pattern Recognition and Machine Learning.* Springer.

Breiman , L. (2001). Random Forests. *Machine Learning 45*, S. 5-32.

Breiman, L., & Cutler, A. (kein Datum). *Random Forests*. Von http://stat-www.berkeley.edu/users/breiman/RandomForests/cc\_home.htm abgerufen

Case Western University. (kein Datum). *Case Western University Bearing Data Center*. Von http://csegroups.case.edu/bearingdatacenter/pages/welcome-case-western-reserve-university-bearing-data-center-website abgerufen

Domingos, P. (October 2012). A Few Useful Things You Need To Know About Machine Learning. *Communications of the ACM Volume 55 Issue 10*, S. 78-87.

Ellis, D. P. (2005). *PLP. RASTA,MFCC and inversion in MATLAB*. Abgerufen am 2013 von http://www.ee.columbia.edu/~dpwe/resources/matlab/rastamat/

Esteller, R., Vachtsevanos, G., Echauz, J., & Litt, B. (February 2001). A Comparison of Waveform Fractal Dimension Algorithms. *IEEE Transactions On Circuits And Systems-I: Fundamental Theory and Applications, Vol. 48, No.2,*.

Kolerus, J., & Wassermann, J. (2008). *Zustandsüberwachung von Maschinen .* Wien: Expert Verlag.

Li, B., Chow, M. Y., Tipsuwan, Y., & Hung, J. C. (October 2000). Neural-Network-Based Motor Rolling Bearing Fault Diagnosis. *IEEE Transactions on Industrial Electronics*.

Liaw, A., & Wiener, M. (2002). *Classification and Regression by randomForest*. Von http://CRAN.R-project.org/doc/Rnews/ abgerufen

Lou, X., Loparo, K. A., Discenzo, F. M., Yoo, J., & Twarowski, A. (2004). A model-based technique for rolling element bearing fault detection. *Mechanical Systems and Signal Processing*, S. pp.1077-1095.

Mandelbrot, B. B. (2004). *Fractals And Chaos.* Berlin: Springer.

Marwala, T. (2012). *Condition Monitoring Using Computational Intelligence Methods.* Springer.

Myers, C., Japkowicz, N., & Gluck, M. (1995). A Novelty Detection approach to classification. *Proceedings Of The Fourteenth International Joint Conference On Artificial Intelligence*, S. 518-523.

Nelvamondo, F. V., Marwala, T., & Mahola, U. (December 2006). Early Classification of Bearing Faults using Hidden Markov Models, Gaussian Mixture Models, Mel Frequency Cepstral Coefficients and Fractals. *International Journal of Innovative Computing, Information and Control*.

Polychronaki, G. E., Ktonas, P. Y., Gatzonis, S., Siatouni, A., Asvestas, P. A., Tsekou, H., et al. (23. June 2010). Comparison of fractal dimension estimation algorithms for epileptic seizure onset detection. *Journal Of Neural Engineering 7*, S. 18pp.

Quinlan, R., & Kohavi , R. (1999). *Decision Tree Discovery.* Stanford.

Raghavendra, B. S., & Dutt, D. N. (2010). Computing Fractal Dimension of Signals using Multiresolution Box-counting Method. *International Journal of Information and Mathematical Sciences 6:1*.

Tax, D. M. (May 2012). DDtools, the Data Description Toolbox for Matlab. S. Version 1.9.1.

Tax, D. M., & Duin, R. P. (January 2004). Support Vector Data Description. *Machine Learning, Volume 54, Issue 1*, S. 54-66.

Tax, M. J. (2001). One Class Classification . *Concept Learning in the absence of counter-examples (Ph.D. Thesis)*. Delft.

van der Heijden, F., Duin, R. P., de Ridder, D., & Tax, D. M. (2004). *Classification, Parameter Estimation and State Estimation.* 2005: Wiley.

Ypma, A., & Duin, P. W. (1998). *Support Objects for Domain Approximation.* Delft: Faculty of Applied Sciences .