Contents

[Notation 2](#_Toc360648423)

[1 Introduction 2](#_Toc360648424)

[1.1 Theory of condition monitoring 2](#_Toc360648425)

[1.2 Overview of this thesis 4](#_Toc360648426)

[2 Feature Extraction 5](#_Toc360648427)

[2.1 Mel Frequency Cepstral Coefficients 6](#_Toc360648428)

[2.2 Multi Fractal Dimensions 7](#_Toc360648429)

[2.3 Kurtosis 8](#_Toc360648430)

[3 Semi-supervised classification 9](#_Toc360648431)

[3.1 Support Vector Data Description 12](#_Toc360648432)

[3.2 K-Center 16](#_Toc360648433)

[3.3 K-Means Data Description 17](#_Toc360648434)

[3.4 Nearest Neighbor Data Description 17](#_Toc360648435)

[3.5 Parzen Window Data Description 18](#_Toc360648436)

[3.6 Self Organizing Maps 18](#_Toc360648437)

[3.7 Random forest 20](#_Toc360648438)

[3.7.1 Variable importance 21](#_Toc360648439)

[3.7.2 Proximity 21](#_Toc360648440)

[3.7.3 Class prototypes 22](#_Toc360648441)

[3.7.4 Outlier measure 22](#_Toc360648442)

[3.7.5 Unsupervised training 22](#_Toc360648443)

[3.7.6 Semi-supervised classification 23](#_Toc360648444)

[4 Experiments 23](#_Toc360648445)

[4.1 Roller Bearing Datasets 25](#_Toc360648446)

[4.2 Feature Extraction 27](#_Toc360648447)

[4.2.1 Mel Frequency Cepstral Coefficients 27](#_Toc360648448)

[4.2.2 Higuchi Fractal Dimensions 28](#_Toc360648449)

[4.3 Random Forest Data Description 29](#_Toc360648450)

[4.4 Classification and Evaluation 31](#_Toc360648451)

[4.4.1 Performance evaluation 31](#_Toc360648452)

[4.4.2 Feature reduction experiment 33](#_Toc360648453)

[5 Conclusions 34](#_Toc360648454)

[6 References 35](#_Toc360648455)

# Notation

# Introduction

## Theory of 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 (*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 of the form

Where is the nth *feature* vector in a set of *N* sample vectors and is the label of a known machine condition applied to . The generation of sample data sets typically requires the involvement of a supervisor, with the expert knowledge to apply meaningful labels to experimental sample data. Techniques utilizing such datasets to construct a model of machine conditions are therefore also 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 all other possible conditions. *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, 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 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 a datasets. *Unsupervised* methods are.

## Overview of this thesis

*Roller bearing* datahas already been used in several 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 Vector* *Machines*. Goal of this thesis is the design and evaluation of an exemplary *semi-supervised* condition monitoring approach, using the roller bearing data collection from (Case Western University) as benchmark.

My contribution involves

* Research in the field of *condition monitoring* and concise presentation of a general condition monitoring approach (1.1)
* Selection of a suitable benchmark data set (4.1)
* Research in *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 in the field of  *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 *one-class-classifier* based on *random forest*
* 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 and discusses 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 and evaluated 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 (4.4.1) compares the classifier performances, while the second analyzes applicability of *random forest* for feature reduction purposes.

# Feature Extraction

Analysis of vibration data measured during machine runtime is the most popular approach to *Machine Condition Monitoring* (Nelvamondo, Marwala, & Mahola, 2006). Consequently, many different methods have been applied to analyze machine vibration signal. Some of these methods, such as the peak level, crest factor or kurtosis analysis use the time signal directly to measure certain aspects of the data. Other methods, such as the *Fourier* or the *Wavelet Transform*, transform the time signal into another domain to expose meaningful data aspects hidden in the time signal.

In this thesis, a mixture of several techniques is used to extract features that can be used to distinguish between normal and faulty machine conditions. Two of these techniques, *kurtosis* and *Multi Fractal Dimensions*  process the time signal directly. The third method, *Mel Frequency Cepstral Coefficients* transforms the time signal into a time-frequency-domain to extract features.

## Mel Frequency Cepstral Coefficients

Mel Frequency Cepstral Analysis is a technique commonly used in Speech Recognition, which can capture the dynamic characteristics of a signal by extracting both linear- and non-linear features (Marwala, 2012) .

The 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 signal.

Some insight into the general 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 is then 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) is transformed into the multiplication

(2.1.3)

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

(2.1.4)

From (2.1.1) and (2.1.4) it can be seen, that a Cepstrum separates the input signal and the impulse response in the time domain, which can be very useful when time signals are compared which are generated by different input signals in combination with the same system characterized by .

Applied to the Condition Monitoring of machines, in (4.1.2) represents the signal measured by accelerometers at a certain point outside the machine. The actual information about the machine condition is contained in the signal which is generated somewhere inside the machine and filtered by the transmission path in the machine. The Cepstrum separates the condition signal from the transmission path (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 the Frequency Spectrum and can be calculated by

1. Transforming the input signal from the time domain to the time-frequency domain through a windowed Fast Fourier Transform:

Where is the number of windows, is the time signal in window n and is the Hamming Window given by

with and the normalization factor .

1. Changing the frequency spectrum to the Mel scale through the equation
2. Converting the logarithmic Mel Spectrum back to the time domain with the Discrete Cosine Transform

## Multi 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 for the calculation of the fractal dimension of time series, often based on some kind of length or distance measurements between the discrete points of a curve over several scales. Popular algorithms involve the Box-counting Method, Katz’s Method and Higuchi’s Method (Raghavendra & Dutt, 2010).

Fractal Dimension Measurement has been successfully used for the extraction of features from machine-vibration data (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 k new time series as

for

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

1. Computing the average length for each of the new time series as

for several scaling factors ranging from 1 to a freely chosen , with and the normalization factor

1. Calculating the complete length 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, which can be calculated by

Where N is the number of samples and the normalization factor is the squared second order moment, .

Kurtosis as quantification of peak sharpness can be useful in Condition monitoring of machines, where faults often manifest themselves in significant changes in the sharpness or spiking of the vibration signal (Nelvamondo, Marwala, & Mahola, 2006).

# Semi-supervised classification

A classifier can be represented by a function

Where is the data vector to be classified, is a vector of parameters which are inferred from labeled samples (training data set) and y is the estimated class for**.**

The optimization of the parameters is often done by minimizing some kind of error function such as the *Sum of Squares Error*, which is given by

where is the class label of the nth training sample vector .

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

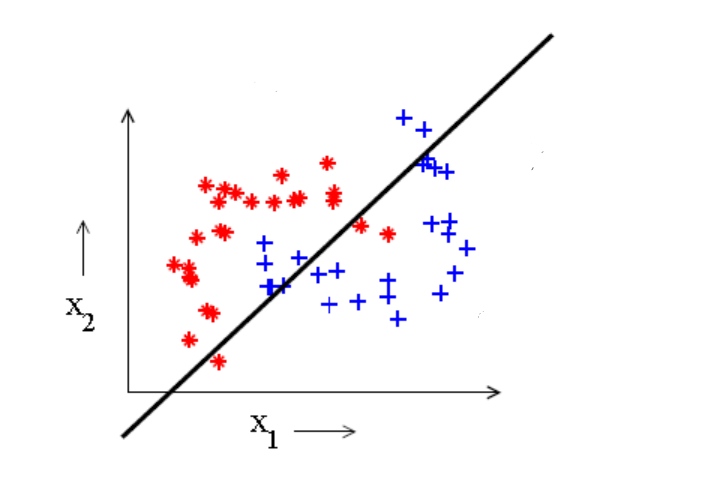


Fig. 2: Two-Class-Classification

The quality of a trained *classifier* is measured by its ability to correctly classify unseen 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 the performance of a supervised classifier is the quality of the sample set used for 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 often referred to as *bias-variance trade-off* (Domingos, 2012).

If sample data is available for only one of the possible classes, 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 target class. New data can then be compared to this *data description* and accepted as target class member if it matches the model closely enough, or rejected as outlier otherwise. Figure 3 illustrates this concept with a closed boundary as *data description* of the target training objects.

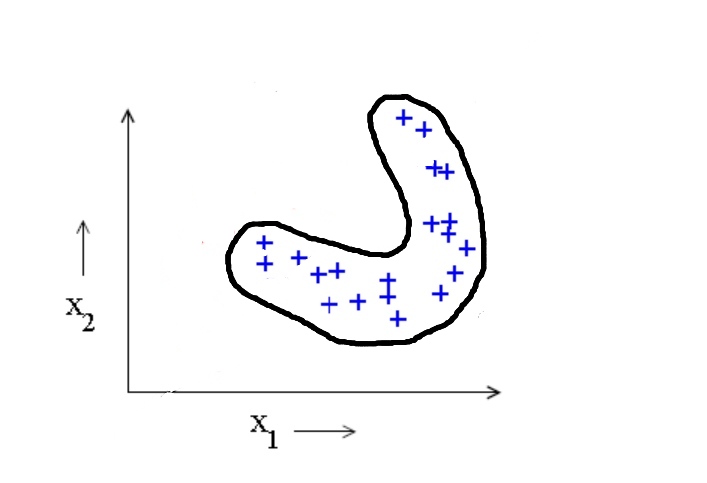


Fig. 3: One-Class-Classifier

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

Were label is applied to data objects which are estimated to belong to the target class and the label to all others. Depending on the context, this classification approach is also known as *Data Description*, *Novelty* Detection, Outlier *Detection* or *Concept learning*. A comprehensive survey of *One-Class-Classifiers* can be found in (Tax M. J., 2001). A

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

(3.1)

or when their probability to belong to the target dataset is above the threshold:

(3.2)

Where is the indicator function.

A common choice for the distance function is the *Euclidean Distance.*

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

(3.3)

where are the training samples 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 has to be found experimentally. For some robustness against outliers in the dataset, should be smaller than one. However, the smaller , the larger the classification error on the target objects.

For the evaluation of the *One-Class-Classifier* performance, two types of errors have to be considered. *Error Type I* is defined as

where is the fraction of rejected targets, and *Error Type II* is defined as

Where is the number of accepted outliers.

The *One-Class-Classificers* used in this thesis, are briefly introduced in the following chapters.

## Support Vector Data Description

For a two-class problem with twofeatures, a *Support Vector Machine* finds an optimal class boundary by identifying the points closest to the class boundary and maximizing their perpendicular distance (also known as *margin*) to the class boundary. This principle is illustrated in figure for a linearly separable two-class training set.



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)

where for all points lying on the plane .

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

(3.1.2)

If (3.1.1) is set to

(3.1.3)

for the *n* training datapointsclosest to the *hyperplane*, then

(3.1.3)

for all training data points and (3.1.2) can be reduced to

which is equivalent to minimizing

(3.1.4)

subject to the constraints given by (3.1.3).

With the introduction of one *Lagrange Multiplier* for each constraintdefined by (3.1.3), the *Lagrangian* is given by

Optimizing the *Lagrangian* with respect to and **,** plugging the results of the optimization into (3.1.1) and with the introduction of a *kernel function*, the *Support Vector* classifier can be formulated as

(3.1.5)

Where training vectors corresponding to are the support vectors and is a *kernel* function (Bishop, 2009). The class membership of an unknown test point is indicated by the sign of (3.1.5).

According to the general definition of *One-Class-Classifiers* given above, the *semi-supervised* variant of a *Support Vector Machine* - also known as *Support Vector Data Description (SVDD)* – tries to find a boundary enclosing all the target training samples. This *boundary* is a *hypersphere* with radius R and center **a,** as illustrated in figur (Tax & Duin, 2004).

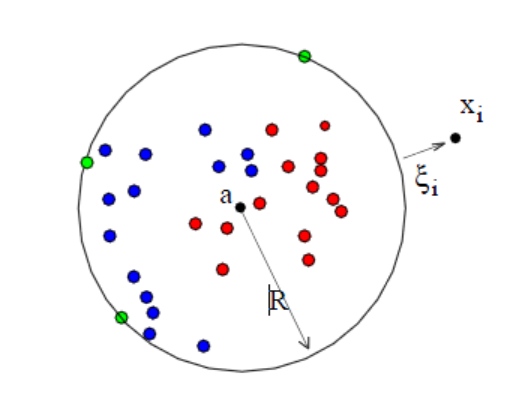


Fig. 5: Support Vector Data Description

The fundamental goal in *SVDD* is the minimization of the radius

such that the *hypersphere* contains all training data points, which is expressed by the constraints

(3.1.5)

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

To deal with the occurrence of outliers in the target training data set as indicated by the point outside the boundary in figure 5, the constraints in (3.1.5) have to be extended by so-called *slack-variables*  to

(3.1.6)

With the new constraintsthe optimization problem changes into

(3.1.7)

where and C controls the tradeoff between a tight decision boundary and the influence of outliers.

A compact formulation of the optimization problem is obtained by combining the objective function(3.1.7) with the corresponding constraints (3.1.6) to the Lagrangian

(3.1.8)

Where and are the Lagrange Multipliers.

By a sequence of steps (Tax M. J., 2001), the final *SVDD* classifier can be derived as

(3.1.9)

Where is the test vector to be classified, and are the nth and the mth training vectors and , are Lagrange multipliers which are only nonzero, if they correspond to support vectors or .

The vector products in (3.1.9) can again be replaced by kernels, resulting in the classifier

Figures 4 to 6 show SVDD boundaries with a Radial Basis Function Kernel and different values for sigma, fitted to the same banana shaped target data set.



Fig. 6 SVDD with RBF, sigma=2



Fig. 7, SVDD with RBF, sigma=3



Fig. 8, SVDD with RBF, sigma=5

## 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 of the dataset ***X***lies in the receptive field of a support object if

And

where *d* can be any kind of distance measure.

The optimization problem of finding a optimal subset with a minimal radius *r* such that all data samples in ***X*** are covered by the receptive field of one support object, can be solved with a variant of the *k-means* algorithm known as *k-*centers. *K-centers* finds a subset such that the maximum distance of all minimum distances between dataset objects and support objects is minimized (Tax M. J., 2001). Thus, the goal of *K-Centers* optimization is the minimization of the error function

With the trainedsupport objects , the distance of a test object to the target set can be calculated by

## 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 their 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

(5.4.1)

In *k-Means* (Tax M. J., 2001).

The minimization of the Error (5.4.1) 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 containing the training data objects closest to the prototype . Then for each cluster, a new prototype is calculated by simply taking the mean of the data points in that cluster. This procedure is reiterated until the prototypes are stable.

After training of the *prototypes*, new objects can be compared to the target class description by simply calculating their *Euclidean Distance* to the closest prototype. The classifier accepts new objects if the distance to their prototype is below the threshold value and rejects them otherwise:

## 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 is the volume of a small sphere centered on a data vector **.** In the supervised case, a test value is classified by increasing until the sphere contains exactly data points 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 training 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

## 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, is the kernel width and is a kernel function which has to satisfy the constraints

And

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

Where h is the standard deviation of the Gaussian components.

*Parzen Window* can be used directly for *One Class Classification*. The only parameter *h* is trained through a maximum likelihood method (Tax M. J., 2001).

## Self Organizing Maps

A *Self Organizing Map* consists of several nodes or neurons. Each node has a defined position in an *N-*dimensional grid structure and is associated with a *D-dimensional* prototype vector, representing *D-*dimensional training data objects. *N* is often chosen to be 1 or 2, such that

The position of a neuron in the *N-dimensional* grid – or feature space – reflects the neighborhood relations in the *D-*dimensional data space, such that neighboring nodes in the grid are associated with neighboring prototypes in data space. Through this topology preserving property, *SOM*s create a mapping from a high dimensional data space to a low dimensional feature space, and allow the visualization of relations from a high dimensional data space (van der Heijden, Duin, de Ridder, & Tax, 2004).

The training starts with an initialization of the prototypes, e.g. by randomly assigning a training data object to each node. Then, for each training data object the following two steps are repeated:

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

Where the subscript *j* denotes the number of the node and the superscript *i* specifies the iteration step in the training sequence.

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 is updated. The input to this weighting function is the distance between the winning neuron determined in step one and neuron which 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 neurons. 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.

## Random forest

*Random Forests* (Breiman , Random Forests, 2001)is an ensemble learning method for classification and regression which grows many trees from sampled subsets of a training set. New objects are run down all trees and labeled with the class that a majority of the trees voted for.

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

1. If the number of cases in the original training set is *N*, *N* cases are sampled from the training set, with replacement.
2. If the number of attributes in the training set is *M*, a number is chosen. For each node in the tree m 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

(5.2.1)

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

(5.2.2)

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 cases, which can be used to cluster date 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
* Extensibility to unlabeled data through the two previous features

Some of the features are a result of the basic tree growing process, while others are based on optional computations and data structures which can easily be integrated into the tree construction.

For the construction of each tree in a random forest, a training set of N cases is uniformly sampled from the original training set with replacement. Thus, a certain number of training number 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 training data objects which were not used in its construction are put down that tree to get a classification.
2. Each training data 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 training samples where the labels applied in step 2 did not match the real classes by the complete number of training 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 a *proximity matrix* which is an matrix, where is the number of training cases used in building the forest and each entry is a measure for the similarity between cases 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 detection of outliers.

### 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 based on the proximity measures.
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 *x* 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

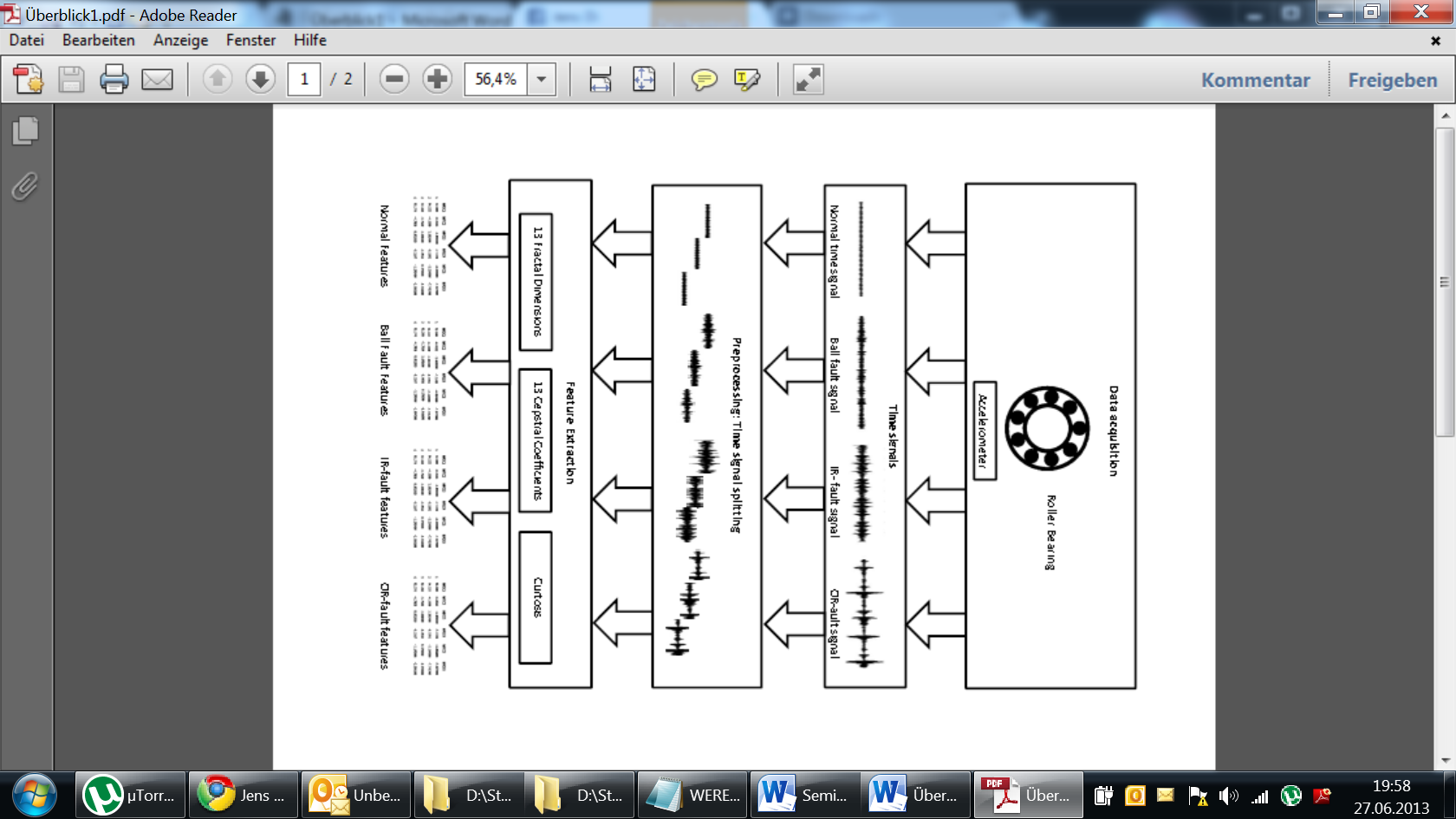
In case of unlabeled training data, the random forest method considers all training data to belong to class 1 and creates a synthetic dataset of the same size and dimension as the training data set, which is labeled as class 2. A synthetic data object of class 2 is created by uniformly sampling from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

The original training data and the synthetic data can then be combined and used to construct a two-class random forest with most of the original options such as the *Proximity Matrix,* *Outlier Detection* and *Variable Importance*.

### Semi-supervised classification

# Experiments

Figure 9 provides an overview of the steps involved in the classification and evaluation approach of this thesis.



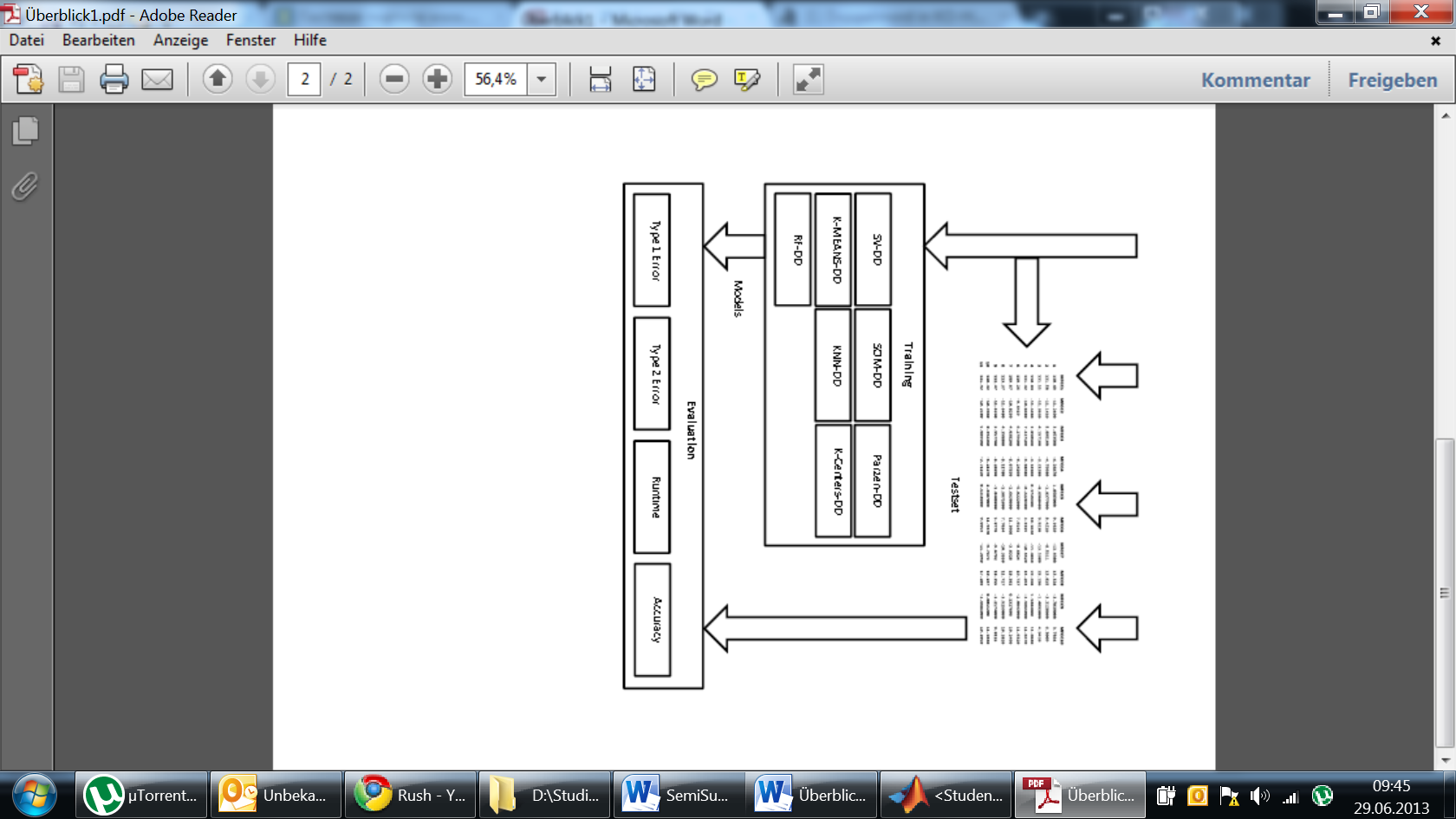


Fig. 9 Experimental Sequence

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 12.

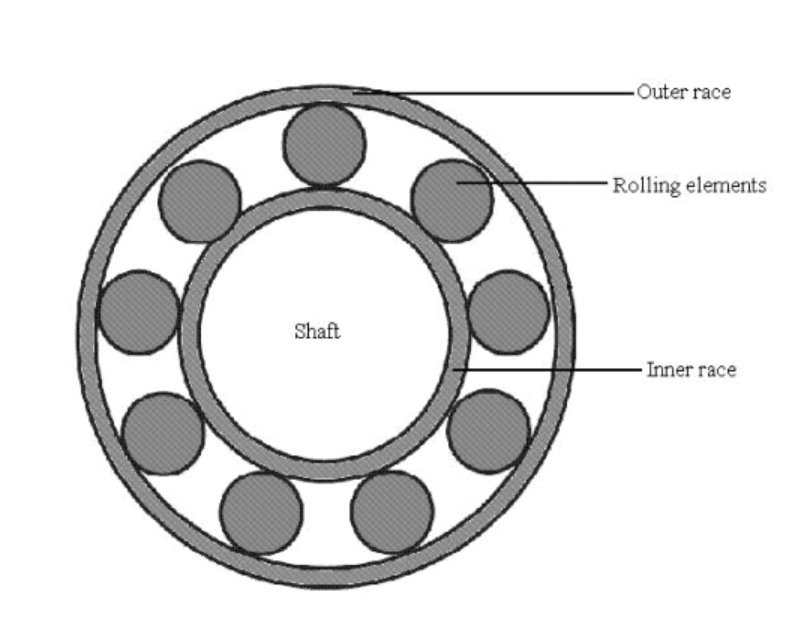


Fig. 10 Roller Bearing

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 3 displays sequences of each data set corresponding to the first five revolutions of the roller bearing.

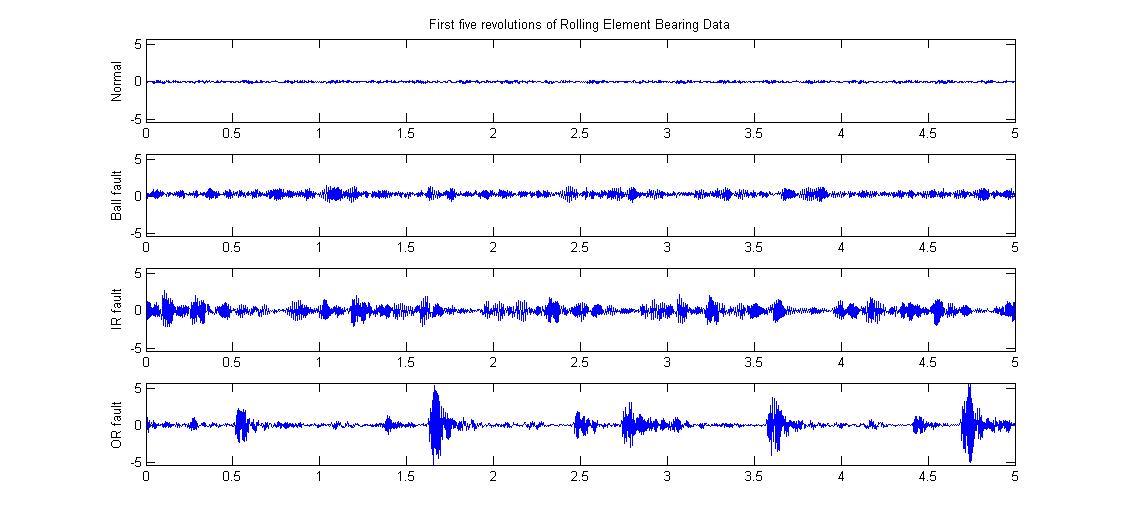


Fig. 11 Roller Bearing Sample Data

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

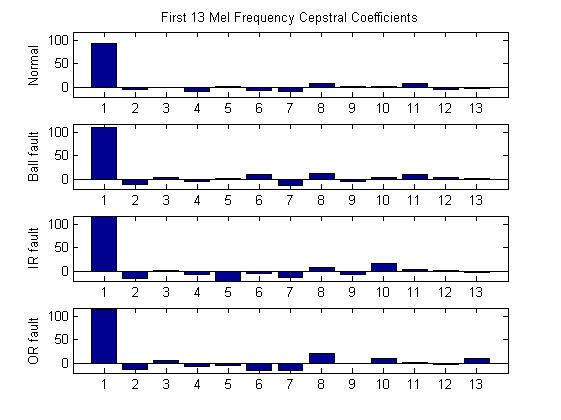


Fig. 12 Thirteen Mel Frequency Cepstral Coefficients

### 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 12 illustrates *HFD* values of the first 15 feature vectors of each set, with .

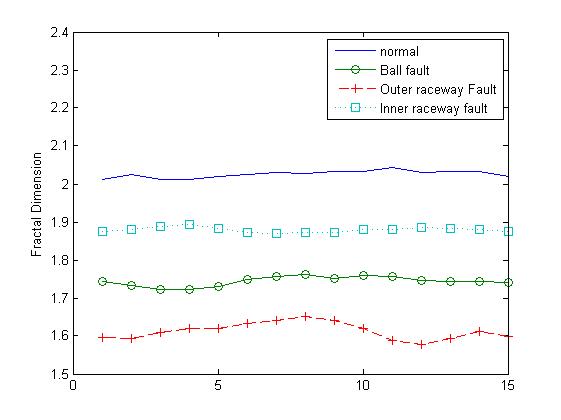


Fig. 13: 15 Higuchi Fractal Dimensions with k=6

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 13 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. 14 shows a plot of the *Outlier Measures* for a *Random Forest* trained with 427 roller bearing normal feature vectors.



Fig. 14 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 15 shows the *variable importance* measures calculated during a *random forest* training with 427 normal feature samples.



Fig. 15 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 16 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. 16 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 Experimental results

Figures 16 and 17 compare the time performances and accuracies of the classifiers.

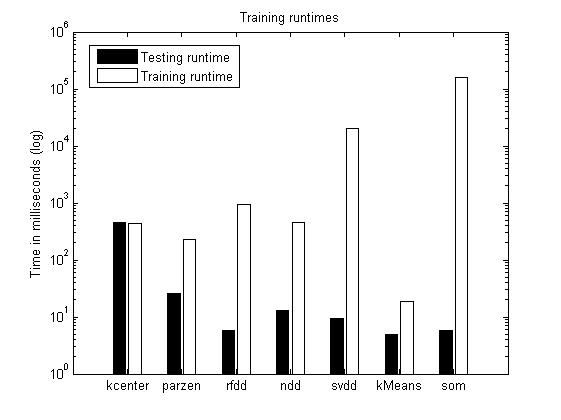


Fig. 17 Time Performance

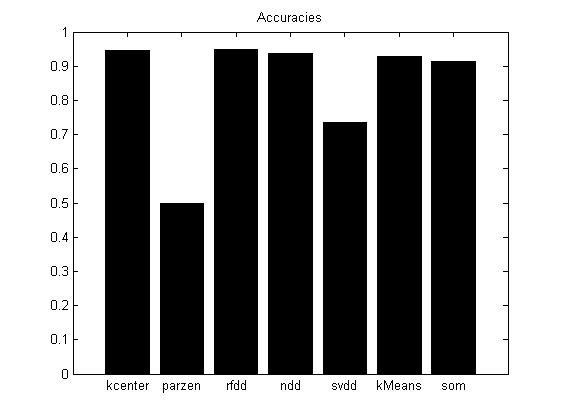


Fig. Accuracies

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 19.

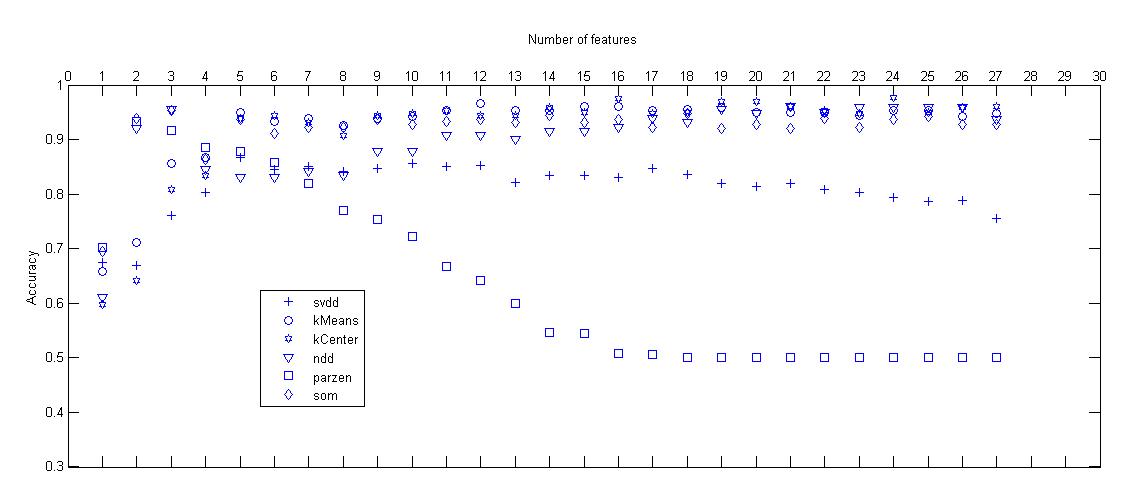


Fig. 19 Accuracies with reduced features

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 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

# 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 .