Outlier Detection in Condition Monitoring

Masterarbeit

zur Erlangung des Grades eines

Master of Science

im Studiengang „Intelligente Eingebettete Mikrosysteme“

vorgelegt von

Ralph Fehrer

aus Karlsruhe

Albert-Ludwigs-Universität

Freiburg im Breisgau

2013

DECLARATION

I hereby declare, that I am the sole author and composer of my Thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.

I hereby also declare, that my Thesis has not been prepared for another examination

or assignment, either wholly or excerpts thereof.

Karlsruhe, 30.07.2013 _____

Place, date Signature

Contents

Notation

1 Introduction

1.1 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 operating 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).

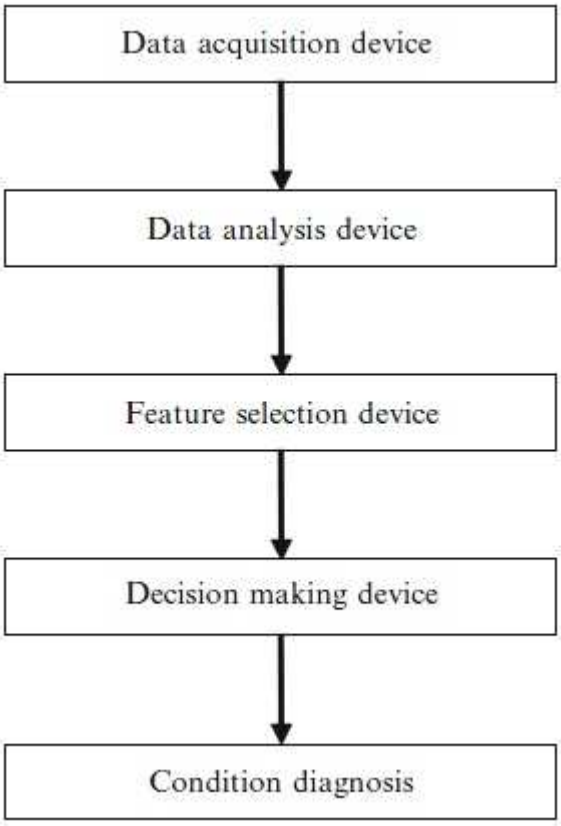


Fig. 1 Condition Monitoring Framework

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 of the form

$$S_N = \{(y_1, t_1), (y_2, t_2), \dots, (y_N, t_N)\}$$

Where N denotes the number of pairs in the signal sequence and y_i represents a sensor value at time t_i.

Purpose of the data analysis step is the extraction of 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* transformations (e.g. *Fourier transform*)
- *Time-frequency domain* transformations, such as the *wavelet transform*, the *short time Fourier transform* (STFT) or the *cepstrum* analysis

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

$$\boldsymbol{x_n} = (x_1, x_2, \dots, x_D)$$

where *n* denotes a specific data vector in a sequence of *N* vectors, $x_1,\dots,x_D$ 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 decision algorithms, of which many are susceptible to the *curse of dimensionality* (Bishop, 2009). Various *feature reduction* techniques can be applied 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 methods or combinations of several techniques, also known as *ensemble methods* (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 type

$$(\boldsymbol{x}_n, y_n)$$

(1.1)

where $\boldsymbol{x_n}$ is the nth *feature* vector in a set of *N* vectors and y_n denotes a known condition *label* linked to the values of $\boldsymbol{x_n}$. Techniques utilizing sample data to construct a model of machine conditions are known as *supervised* methods. If sample data is available for all possible machine conditions, supervised algorithms can be trained to distinguish conditions and label new *feature* vectors accordingly. A special situation arises if representative sample data is available for only one condition. 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 required the destruction of a machine in all possible ways (Myers, Japkowicz, & Gluck, 1995). In such scenarios, special algorithms can be applied to create a description of the known condition and to decide whether a new data object matches that description or not.

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* sequence, 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 at this level are usually made by experts (Marwala, 2012).

1.2 Overview

Analysis of vibration signals from *roller bearing* elements is a standard problem in machine condition monitoring, since many machine failures are linked to bearing failures (Nelvamondo, Marwala, & Mahola, 2006). Consequently, roller bearing signals have been used in many publications, to develop and evaluate *condition monitoring* techniques (Marwala, 2012), (Nelvamondo, Marwala, & Mahola, 2006), (Li, Chow, Tipsuwan, & Hung, 2000). In all of these papers, both normal and fault condition signals were used to train *supervised* classifiers such as *neural networks* or *support vector machines*.

Goal of this Thesis is the presentation of an exemplary *outlier detection* approach using the *roller bearing* dataset from (Case Western University) as benchmark. In this approach, as opposed to the papers mentioned above, only the normal condition samples are used to train a special type of *classifiers*, known as *one-class-classifiers.* The available fault condition samples are then used for the evaluation of these classifiers.

My work involves the following aspects:

- 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 dataset (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 dataset and the extracted *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 *outlier detection* approach, including *feature extraction*, *feature selection*, classification and evaluation (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 *one-class* 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 (3.7.6), (4.3).

The second part of this Thesis (4) combines and evaluates the techniques introduced in the first part and starts with a description of the *roller bearing* benchmark set in 4.1. The *feature*

extraction results are briefly discussed in 4.2 and 4.3 provides a proof of concept of the *one-class 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 evaluates the applicability of the *random forest variable importance* for feature reduction purposes.

2 Feature extraction

Vibration analysis is the most widely used 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 original 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.

2.1 Mel frequency cepstral coefficients

Mel frequency cepstral 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 a measured frequency spectrum to the characteristics the human auditory system.

The basic *cepstrum* $C(\tau)$ is defined as the inverse *Fourier transform* of a logarithmic frequency spectrum:

$$C(\tau) = \mathcal{F}^{-1}\left(\log\left(\mathcal{F}\left(f(t)\right)\right)\right)$$

(2.1.1)

Where $\mathcal{F}^{-1}$ is the inverse Fourier Transform and $\mathcal{F}(f(t))$ is the frequency spectrum of a time signal $f(t)$.

Insight into a fundamental property of a *cepstrum* can be gained by regarding a time signal $y(t)$ as the output of a *linear time invariant (LTI)* system, characterized by its *impulse response* $h(t)$. The output $y(t)$ of an *LTI* is given by the convolution of an input signal $x(t)$ and the impulse response

$$y(t) = x(t) * h(t).$$

(2.1.2)

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

$$\mathcal{F}\left(y(t)\right) = \mathcal{F}\left(x(t)\right)\mathcal{F}\left(h(t)\right).$$

(2.1.3)

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

$$C(\tau) = \mathcal{F}^{-1}\left(log\left(\mathcal{F}\left(x(t)\right)\right)\right) + \mathcal{F}^{-1}\left(log\left(\mathcal{F}\left(h(t)\right)\right)\right).$$

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

$$C_y(\tau) = C_x(\tau) + C_H(\tau).$$

(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, $y(t)$ in (2.1.2) represents an output signal measured by accelerometers at certain points outside a machine. The actual signal $x(t)$ is generated somewhere inside the machine and filtered by its transmission path through the machine. The *cepstrum* now separates signal from transmission path characteristics (Kolerus & Wassermann, 2008).

The *MFCC* techniques 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 $x(t)$ from the time domain into the time-frequency domain through a windowed *Fourier transform*:

$$y(m) = \frac{1}{F}\sum_{n=0}^{F-1} x(n)w(n)e^{-j\frac{2\pi}{F}nm}$$

Where F is the number of frames, $0 \leq m \leq (F-1)$, $x(n)$ is the time signal in window n and $w(n)$ is the *hamming window* given by

$$w(n) = \beta\left(0.5 - 0.5cos\frac{2\pi n}{F-1}\right)$$

with $0 \leq n \leq F-1$ and a normalization factor β.

2. Changing the frequency spectrum to the *mel* scale using the equation

$$mel = 2595 \times log_{10}\left(1 + \frac{f_{Hz}}{700}\right)$$

3. Converting the logarithmic *mel* spectrum back to the time domain using the *discrete cosine transform*

$$C_m^i = \sum_{n=0}^{F-1} \cos\left(m\frac{\pi}{F}(n+0.5)\right) log_{10}(H_n)$$

where $0 \leq m \leq (L-1)$, L is the number of frequencies extracted from the i[th] signal frame and H_n is the transfer function of the n[th] filter on the filter bank (Marwala, 2012).

2.2 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 N samples

$$x(1), x(2), \dots, x(N)$$

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

1. Constructing k new time series as

$$x_m^k = \left\{ x(m), x(m+k), x(m+2k), \dots, x\left(m + \left\lfloor \frac{N-m}{k} \right\rfloor k\right) \right\},$$

$$\text{for } m = 1,2 \dots, k$$

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

2. Computing the average length for each of the new curve x_m^k as

$$L_m(k) = \frac{\sum_{i=1}^{\left\lfloor \frac{N-m}{k} \right\rfloor} |x(m+ik) - x(m+(i-1)k)|(N-1)}{\left\lfloor \frac{N-m}{k} \right\rfloor k}$$

with $m = 1,2 \ldots, k$ and the normalization factor

$$\frac{(N-1)}{\left\lfloor \frac{N-m}{k} \right\rfloor k}.$$

3. Calculating the complete length for each k as sum of the average lengths:

$$L(k) = \sum_{m=1}^{k} L_m(k)$$

4. Estimating the Fractal Dimension *D* as the slope of the least squares linear best fit line for the curve $ln(L(k))$ versus $ln\left(\frac{1}{k}\right)$, since $L(k) \sim k^{-D}$.

2.3 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

$$K = \frac{1}{N} \sum_{i=1}^{N} \frac{(x_i - \mu)^4}{\sigma^4}$$

Where N is the number of samples, μ is the mean and σ 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).

3 One-Class Classification

A *classifier* can be represented by the function

$$y = f(\boldsymbol{x}, \boldsymbol{w})$$

Where $\boldsymbol{x}$ is the data object to be classified, $\boldsymbol{w}$ is a parameter or weight vector and y is the estimated for $\boldsymbol{x}$.

In a supervised setting, the parameters $\boldsymbol{w}$ are 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

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\{f(\boldsymbol{x}_n, \boldsymbol{w}) - y_n\}^2$$

where y_n is the class label of the nth sample object $\boldsymbol{x}_n$.

If training samples are available for all possible classes, a classifier can be trained to distinguish between all possible classes in *feature space*. Figure 2 illustrates a (non-optimal) *class boundary* for a two-class problem in a 2-dimensional *feature space*.

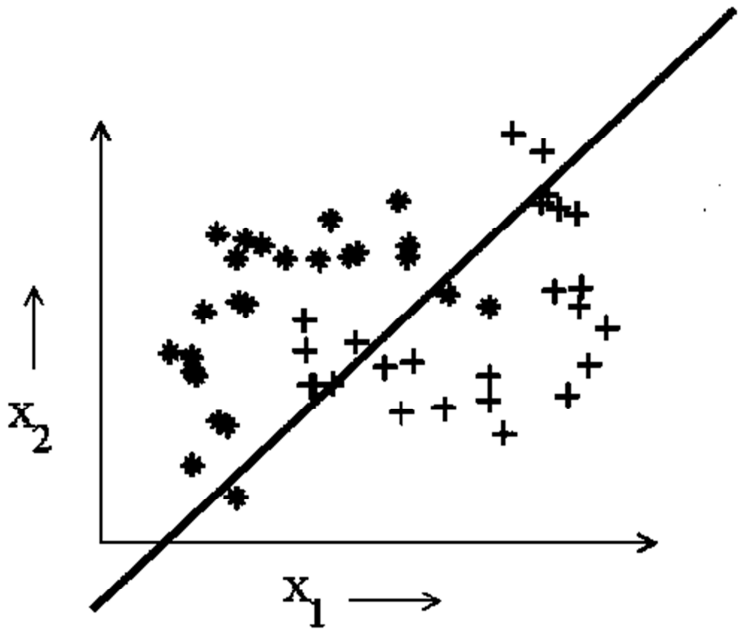


Fig. 2: Two-Class-Classification

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 $\boldsymbol{w}$. The line illustrated in figure 2 is not flexible enough to capture the given class distribution 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 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.

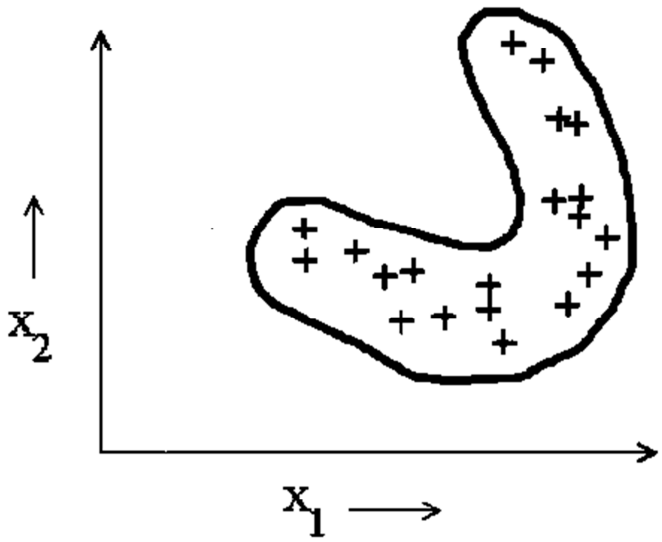


Fig. 3: One-Class-Classifier

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

$$f: \mathbb{R}^D \rightarrow \{1,0\}$$

where label 1 is applied to objects that belong to the *target class* and label 0 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).

Almost 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* sample data. This function can be either some kind of distance measure

$$d(\boldsymbol{x})$$

or a *resemblance probability*

$$p(\boldsymbol{x}).$$

The second component is a threshold θ which defines the decision boundary New objects are accepted as *target class* members if their distance to the *target class* is smaller than the threshold

$$f(\boldsymbol{x}) = I(d(\boldsymbol{x}) < \theta),$$

(3.1)

or if their probability of belonging to the *target* sample set is above the threshold

$$f(\boldsymbol{x}) = I(p(\boldsymbol{x}) > \theta)$$

(3.2)

Where $I(\cdot)$ is the *indicator function*.

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

$$F_{T+} = \frac{1}{N}\sum_{i=1}^{N} I(d(\boldsymbol{x_i}) < \theta)$$

(3.3)

where $\boldsymbol{x}_n$ is the nth training sample and F_{T+} is the fraction of accepted target samples from the training data. For a probability based approach, the threshold is accordingly given by

$$F_{T+} = \frac{1}{N}\sum_{n=1}^{N} I(p(\boldsymbol{x}_n) > \theta)$$

(3.4)

The optimal value for F_{T+} has to be found experimentally. For some robustness against outliers in the training dataset, F_{T+} should be smaller than one. However, the smaller F_{T+}, 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

$$E_1 = F_{T-}$$

where F_{T-} is the fraction of rejected targets, and *error type 2* is defined as

$$E_2 = F_{O+}$$

Where F_{O+} is the number of accepted outliers.

The distance measure $d(\boldsymbol{x})$ and the probability measure $p(\boldsymbol{x})$ 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.*

3.1 Support vector data description

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

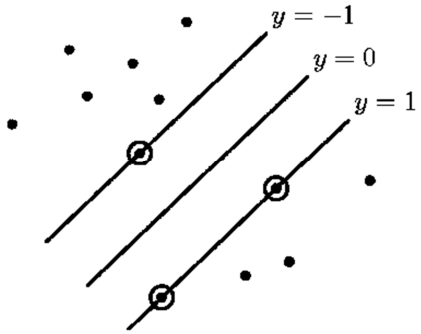


Fig. 4: Support Vector Machine

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

$$\mathrm{y}(\mathbf{x}) = \mathbf{w}^{\mathbf{T}}\mathbf{x} + \mathrm{b}$$

(3.1.1)

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

$$\arg\max_{w,b}\left\{\frac{1}{\|\mathbf{w}\|}\min_{n}[y_n(\mathbf{w}^{\mathbf{T}}\mathbf{x}_n + \mathrm{b})]\right\}.$$

(3.1.2)

Where y_n is the class label of the n[th] training sample $\mathbf{x}_n$.

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

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n y_n k(\mathbf{x}, \mathbf{x}_n) + b$$

(3.1.3)

where the training vectors $\mathbf{x}_n$ that correspond to an $a_n > 0$ are support vectors and $k(\mathbf{x}, \mathbf{x}_n)$ is a *kernel* function (Bishop, 2009). The class membership of an unknown test point **x** is indicated by the sign of (3.1.3).

A *one-class* variant of the *support vector machine* 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.

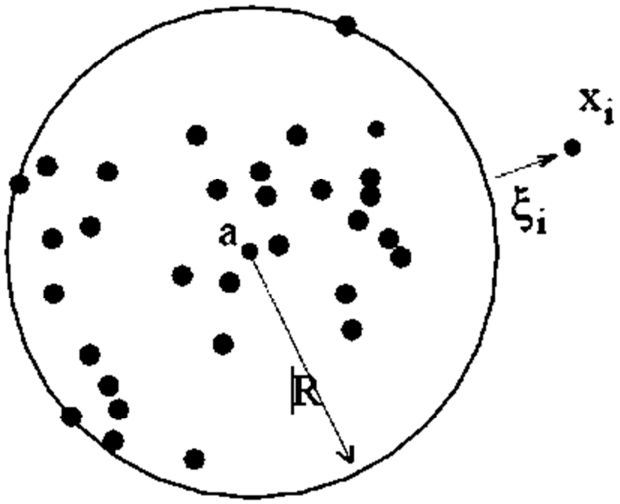


Fig. 5: Support Vector Data Description

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

$$\|\boldsymbol{x}_n - \boldsymbol{a}\|^2 \leq R^2$$

(3.1.4)

Where $\boldsymbol{a}$ and *R* are the center and the radius of the hyperspere and $\boldsymbol{x}_n$ 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* ξ_n, which extends (3.1.4) to

$$\|\boldsymbol{x}_n - \boldsymbol{a}\|^2 \leq R^2 + \xi_n.$$

(3.1.5)

The effect of slack variables is indicated in figure 5 by an outlier $\boldsymbol{x}_i$ and the corresponding slack variable ξ_i.

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

$$f(\boldsymbol{x}, \boldsymbol{\alpha}, R) = I(\|\boldsymbol{x} - \boldsymbol{a}\|^2 \leq R^2) = I\left(\|\mathbf{x}\|^2 - 2\sum_{n=1}^{N} a_n(\mathbf{x} * \mathbf{x_n}) + \sum_{n,m} a_n a_m\,(\mathbf{x_n} * \mathbf{x_m}) \leq R^2\right)$$

(3.1.6)

where $\boldsymbol{x}$ is the test object to be classified, $\mathbf{x_n}$ and $\mathbf{x_m}$ are the nth and the mth training vectors and a_n and a_m are Lagrange multipliers which are only nonzero, if the corresponding $\mathbf{x_n}$ and $\mathbf{x_m}$ are support vectors.

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

$$\begin{aligned} f(\boldsymbol{x}, \boldsymbol{\alpha}, R) &= I(\|\boldsymbol{x} - \boldsymbol{a}\|^2 \leq R^2) \\ &= I\left(\|\mathbf{x}\|^2 - 2\sum_{n=1}^{N} a_n k(\mathbf{x}, \mathbf{x_n}) + \sum_{n,m} a_n a_m\, k(\mathbf{x_n}, \mathbf{x_m}) \leq R^2\right). \end{aligned}$$

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

$$k(\mathbf{x}, \boldsymbol{x}') = exp\left(\frac{-\|\mathbf{x} - \boldsymbol{x}'\|^2}{2\sigma^2}\right).$$

Figure 6 shows an *SVDD* boundary calculated with a *Gaussian kernel* for a banana shaped target data set with 2 features.

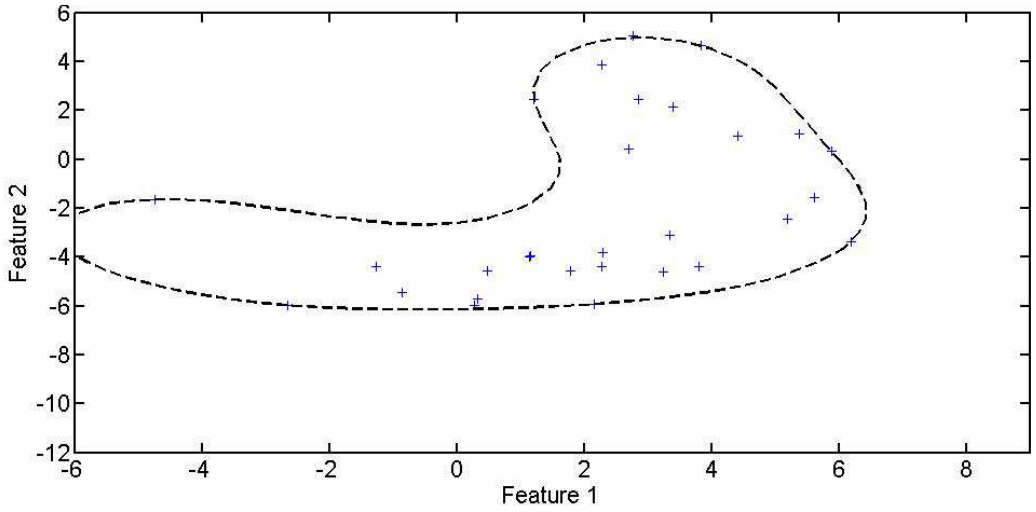


Fig. 6 SVDD with Gausian kernel

3.2 K-center data description

K-centers is based on the *domain approximation* algorithm (Ypma & Duin, 1998), where a subset S_k of k support objects is selected from the N samples of a training dataset $\boldsymbol{X}$ and each of the support objects is surrounded by a *receptive field* of equal radius r. A sample object $\boldsymbol{x}_n$ of the dataset $\boldsymbol{X}$ lies in the *receptive field* of a support object $\boldsymbol{\mu}_p$ if

$$p = arg\min_j d(\boldsymbol{x}_n, \boldsymbol{\mu}_j)$$

and

$$d(\boldsymbol{x}_i, \boldsymbol{\mu}_j) \leq r.$$

I.e., each training object $\boldsymbol{x}_i$ lies in the receptive field of the nearest support object, as long as the distance to that support object is smaller than the radius r.

The problem of finding the subset S_k with the minimal radius r such that all data samples in $\boldsymbol{x}_n$ are covered by the receptive field of one support object is tantamount to minimizing the error

$$E_{k-center} = \max_n \left(\min_k \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2 \right).$$

The optimization algorithm starts with k 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 $E_{k-center}$ is used (Tax M. J., 2001). The parameter k and the maximal number of retries have to be supplied the user.

With the support objects $\boldsymbol{s}_k$ determined in the training phase, the distance of a *test object* $\boldsymbol{x}$ to the target set can be calculated by

$$d_{k-centers}(\boldsymbol{x}) = \min_k \|\boldsymbol{x} - \boldsymbol{\mu}_k\|^2.$$

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.

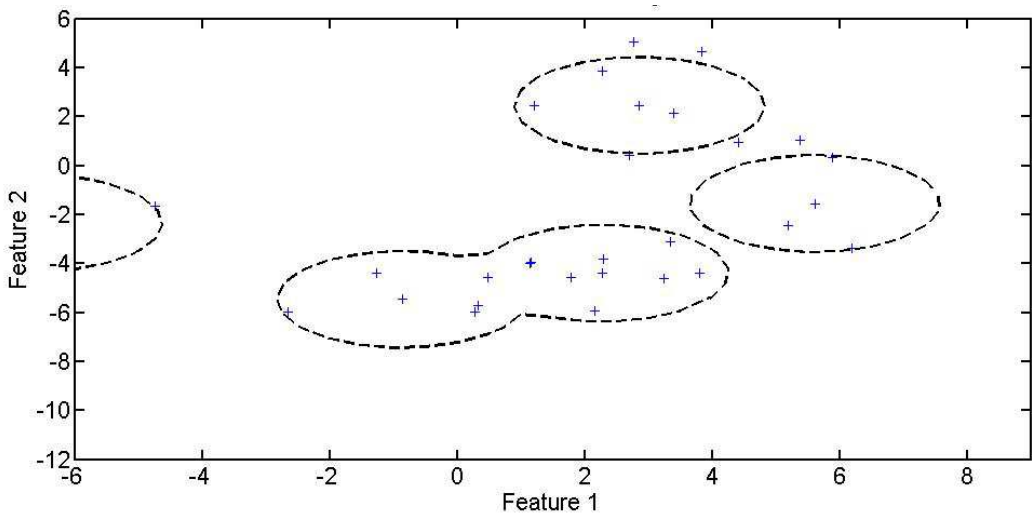


Fig. 7 K-Center boundary

3.3 K-means data description

K-means is a clustering algorithm similar to *k-centers* where *k* cluster centers $\boldsymbol{\mu}_k$ are defined and the *training data* objects belong to the cluster of the nearest cluster center. The main differences to *k-centers* are the placement of the cluster centers - which in *k-means* is not constrained to coincide with existing *training data* objects - and the *error function*, which is defined by

$$E_{k-means} = \sum_{n=1}^{N} \min_k \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2$$

where $\boldsymbol{x}_n$ is the nth of *N* training objects.

The minimization of $E_{k-means}$ implies a placement of the *k* cluster centers such that the distance of each training object to the closest center is a minimum. A typical algorithm for this optimization problem starts with a random placement of *k* cluster centers. Then for each cluster, a new center is calculated by simply taking the mean of the data objects in that cluster. This procedure is reiterated until convergence.

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

$$d_{k-means}(\boldsymbol{x}) = \min_k \|\boldsymbol{x} - \boldsymbol{\mu}_k\|^2.$$

Data objects are labeled as *targets* if this distance is smaller than a predefined threshold or as outliers otherwise

Figure 8 shows a *k-means***Fehler! Textmarke nicht definiert.** boundary for a banana shaped target data set with 2 features.

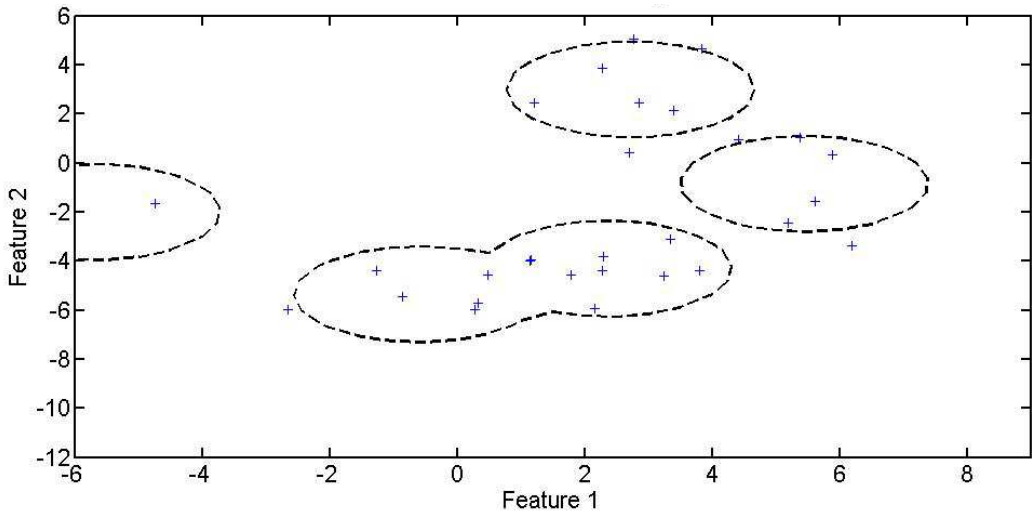


Fig. 8 K-Means-dd boundary

3.4 K –nearest-neighbor data description

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

$$p(\boldsymbol{x}) = \frac{K}{NV}$$

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

In an adaption of the *K-Nearest-Neighbor* method to *one-class* problems (Tax M. J., 2001), the distance of a test object $\boldsymbol{x}$ to its nearest neighbor in the sample set $NN(\boldsymbol{x})$ is compared to the distance between the nearest neighbor of $\boldsymbol{x}$ and the nearest neighbor of the nearest neighbor of $\boldsymbol{x}$, $NN\big(NN(\boldsymbol{x})\big)$. The test object is accepted if the distance $d\big(\boldsymbol{x}, NN(\boldsymbol{x})\big)$ is smaller than or equal to the distance $d\left(NN(\boldsymbol{x}), NN\big(NN(\boldsymbol{x})\big)\right)$.

This can be formalized by the *classifier* function

$$f(\boldsymbol{x}) = I\left(\frac{\|\boldsymbol{x} - NN(\boldsymbol{x})\|}{\|NN(\boldsymbol{x}) - NN\big(NN(\boldsymbol{x})\big)\|} \leq 1\right).$$

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

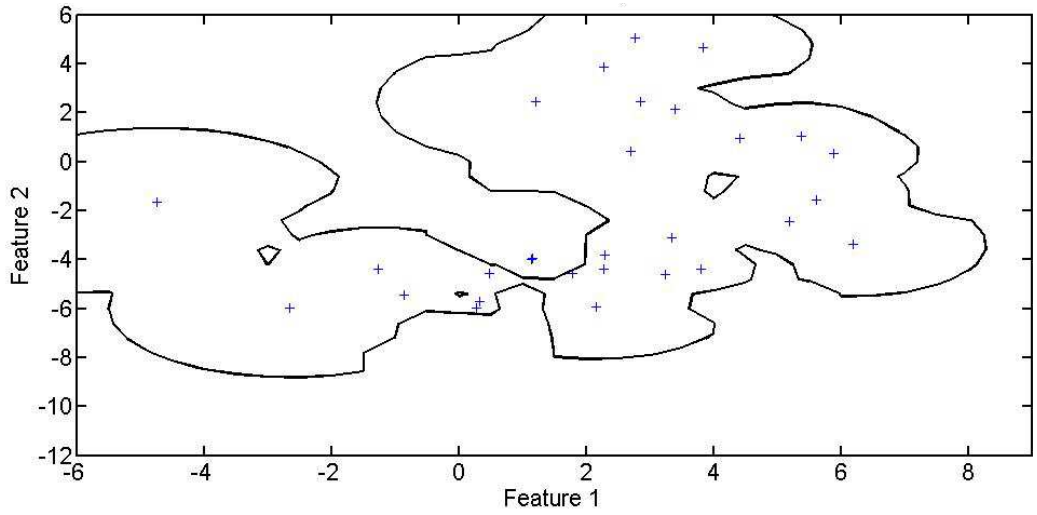


Fig. 9 NN-dd boundary

3.5 Parzen data description

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

$$p(\boldsymbol{x}) = \frac{1}{Nh^D}\sum_{n=1}^{N} k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right)$$

(5.7.1)

where $\boldsymbol{x}_n$ is the n[th] training point, and $k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right)$ is a *kernel function* with the width h, which has to satisfy the constraints

$$k(\boldsymbol{u}) \geq 0$$

and

$$\int k(\boldsymbol{u})d\boldsymbol{u} = 1.$$

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

$$p(\boldsymbol{x}) = \frac{1}{N}\sum_{n=1}^{N} \frac{1}{(2\pi h^2)^{\frac{D}{2}}} exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}_n\|^2}{2h^2}\right).$$

Where h 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.

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

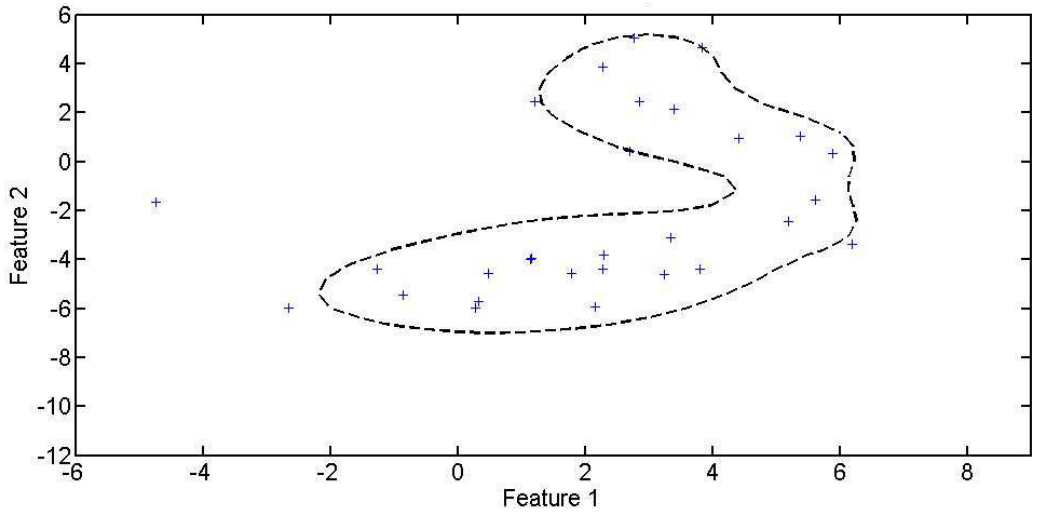


Fig. 10 Parzen-dd boundary

3.6 Self organizing maps data description

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 clusters of *D-dimensional* sample data objects in the *input space*.

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

$$N \ll D.$$

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 *input 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 $\boldsymbol{\omega}_j$ where j denotes the node number and $\boldsymbol{\omega}_j$ is the *prototype* associated with node j. Initialization can simply be done by randomly assigning a training data object as *prototype* to each node.

Then, for each sample data object $\boldsymbol{x}_n$**,** the following two steps are executed:

1. Detection of the node j whose associated *prototype* is closest to $\boldsymbol{x}_n$ by calculating

$$k(\boldsymbol{x}_n) = arg\min_j \left\| \boldsymbol{x}_n - \boldsymbol{\omega}_j^n \right\|$$

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

2. Update of the winning node and its grid neighbors by computing for each node j

$$\boldsymbol{\omega}_j^{(n+1)} = \boldsymbol{\omega}_j^n + \eta^n h^n(|k(\boldsymbol{x}_n) - j|)\left(\boldsymbol{x}_n - \boldsymbol{\omega}_j^n\right)$$

Where η^n is a learning rate which can be set to decline with the iteration count and h^n is a weighting function which determines by how much a node *prototype* is updated. The input to this weighting function $|k(\boldsymbol{x}_n) - j|$ 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

$$h^n(0) = 1$$

and

$$h^n(u) < 1, \forall u \neq 0$$

to ensure that the winning node receives the largest update.

A common choice for the weighting function is the Gaussian function

$$h^n(\boldsymbol{x}) = \exp\left(-\frac{\boldsymbol{x}^2}{\sigma^2_{(n)}}\right)$$

Where $\sigma^2_{(n)}$ defines the width of the neighborhood area

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. Figure 11 shows an *SOM-dd* boundary calculated for a banana shaped dataset with 2 features.

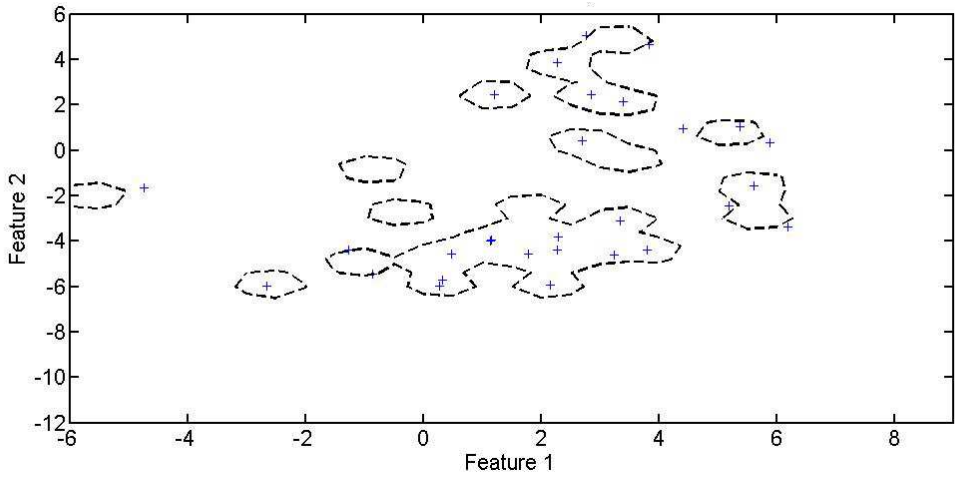


Fig. 11 SOM-dd boundary

3.7 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 $m \ll M$ 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 m resulting in the highest *information gain*

$$G(S,m) = I(S) - \sum_{i=1}^{t} \frac{|S_i|}{|S|} I(S_i)$$

(3.7.1)

Where S is the dataset before splitting, S_i are the splitted datasets and $I(S)$ is the *gini index*

$$I(S) = 1 - \sum_{j=1}^{x} RF\left(C_j, S\right)^2,$$

(3.7.2)

where $RF\left(C_j, S\right)$ is the relative frequency of cases with class j in dataset S.

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.

3.7.1 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

$$\frac{c - p}{N}$$

where N is the number of trees in the forest.

3.7.2 Proximity

One of the most useful features in *random forests* is the *proximity matrix* which is an $N \times N$matrix, where N is the number of data objects used in the forest construction and each entry P_{ij} is a measure for the similarity between objects i and j.

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 i and j in step 1, the value P_{ij} is incremented by 1.

3. After the complete forest is constructed, the entries P_{ij} are normalized by the number of trees in the forest.

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

3.7.3 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.

3.7.4 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 $\boldsymbol{x}$ in class *c* to all a other objects in class *c* is

$$\bar{P}(\boldsymbol{x}) = \sum_{class(k)=c} {P_{x,k}}^2.$$

Based on this average proximity, a raw outlier measure for point $\boldsymbol{x}$ is defined as

$$Outlier_{Raw}(\boldsymbol{x}) = \frac{N}{\bar{P}(\boldsymbol{x})}$$

Where N is the number of samples in the training set.

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

$$Outlier(x) = \frac{Outlier_{Raw}(\boldsymbol{x}) - Median(Outlier_{Raw}(\boldsymbol{x}_i))}{MAD(Outlier_{Raw}(\boldsymbol{x}_i))}$$

Where $Median(Outlier_{Raw}(x_i))$ is the median of all raw outlier measures of objects in class $class(\boldsymbol{x})$ and $MAD(Outlier_{Raw}(\boldsymbol{x}_i))$ is the *Median Absolute Deviation* of all raw outlier measures in class $class(\boldsymbol{x})$.

3.7.5 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*.

3.7.6 Data description

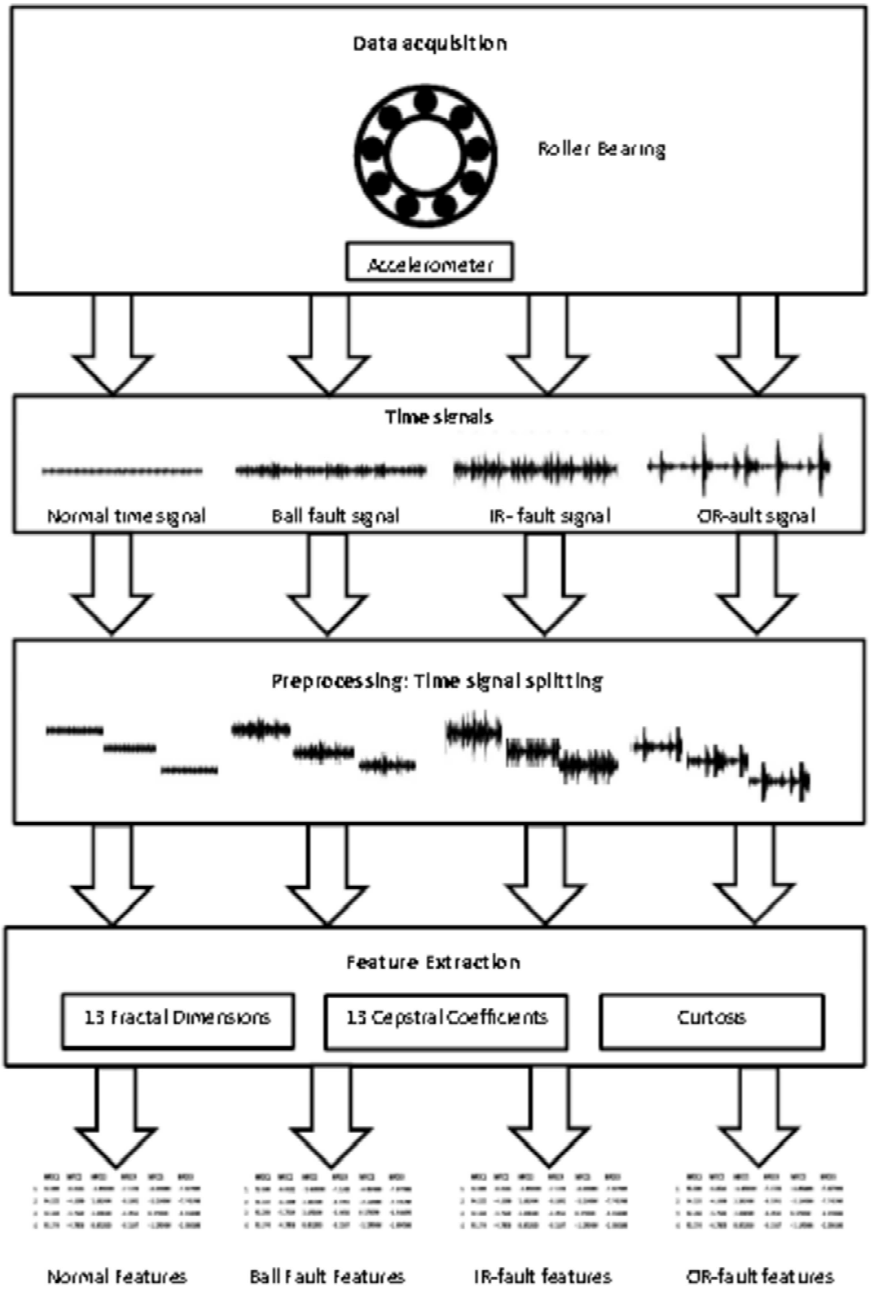
A basic approach for *random forest* based *one-class classification* can be outlined as follows:

- 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 above
- 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.

4 Experiments

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


Data acquisition
Roller Bearing
Accelerometer
Time signals
Normal time signal
Ball fault signal
IR- fault signal
OR-ault signal
Preprocessing: Time signal splitting
Feature Extraction
13 Fractal Dimensions
13 Cepstral Coefficients
Curtosis
Normal Features
Ball Fault Features
IR-fault features
OR-fault features

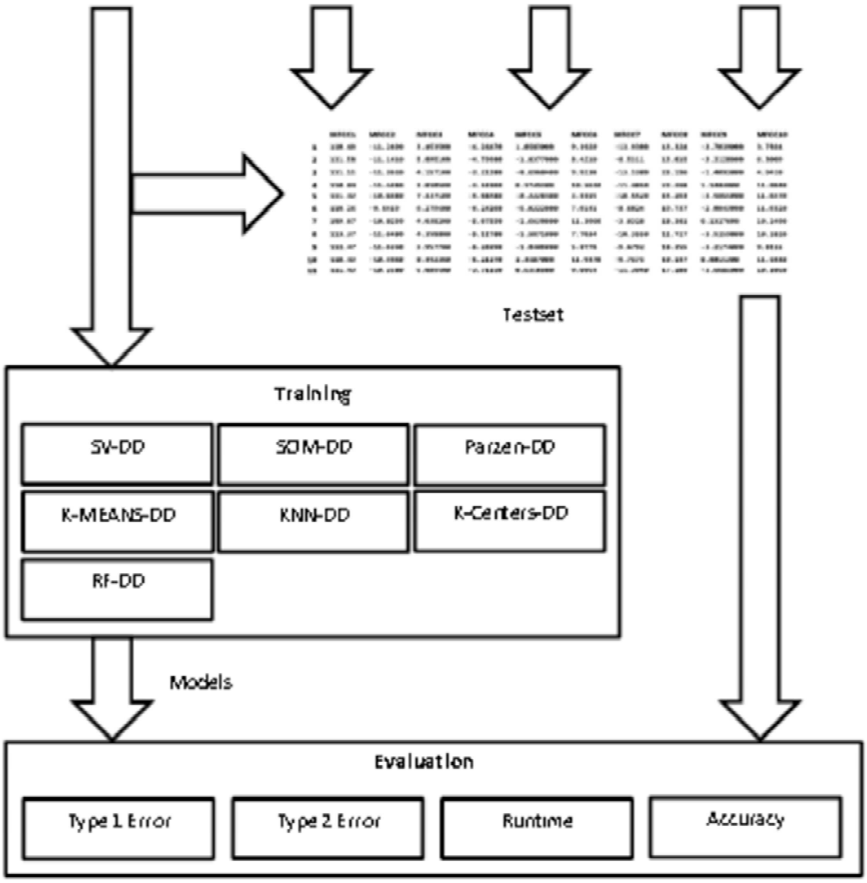


Fig. 12 Experimental Sequence

A brief introduction to the *roller bearing* datasets is given in 4.1. The preprocessing and feature extraction steps are summarized in 4.2 Chapter 4.3 provides an evaluation of the *random forest one-clas classifier* concept and 4.4 summarizes the *outlier detection* and *feature* reduction results.

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* provided by the TU Delft (Tax D. M., 2012). The *random forest* approach introduced in 3.7.6 was evaluated based on an R-package (Liaw & Wiener, 2002) and then integrated into the *Dd_tools* framework.

4.1 Roller bearing datasets

Rotating Machines are very common in various industrial applications. Failures of rotating machines are often linked to *bearing* failures (Nelvamondo, Marwala, & Mahola, 2006). 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 raceway* (IR) and *outer raceway* (*OR*), with a set of rolling elements running between their tracks, as illustrated in Figure 13.

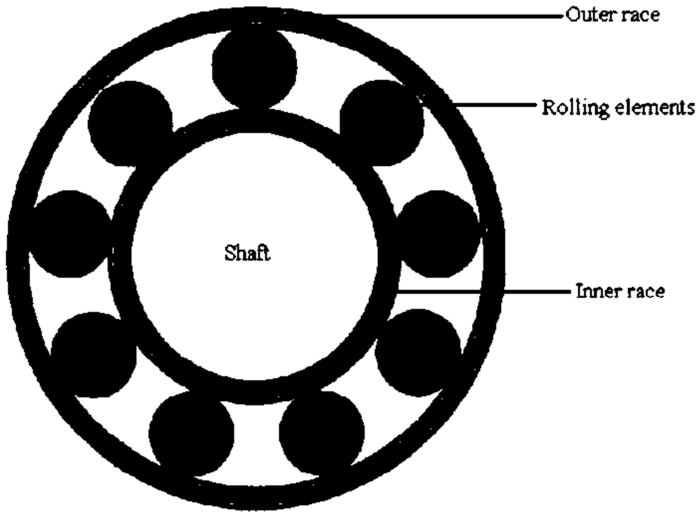


Fig. 13 Roller Bearing

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 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)

For the experiments, for each of the four conditions 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 condition signal corresponding to the first five revolutions of the *roller bearing.*

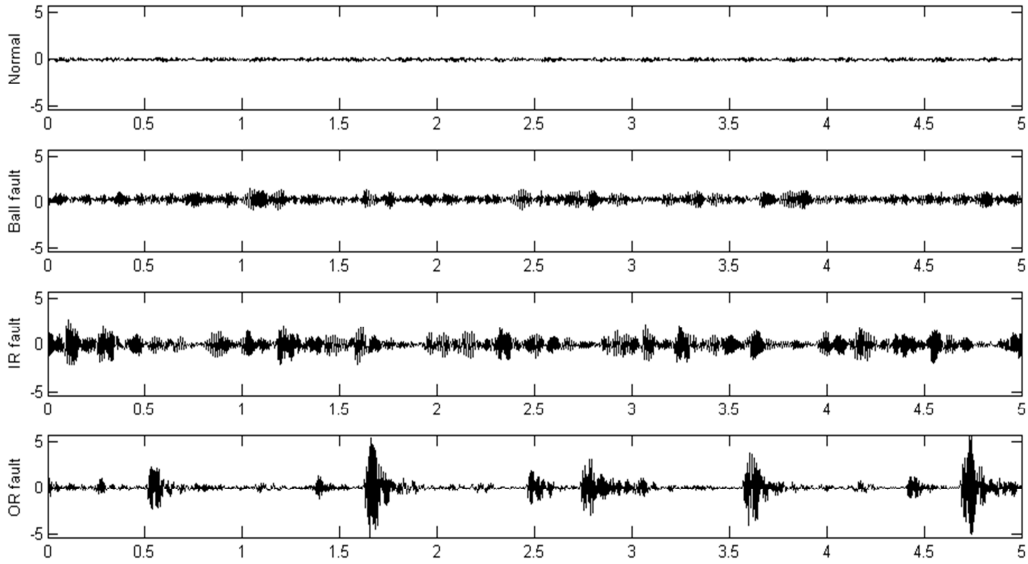


Fig. 14 Roller Bearing Sample Data

To simulate a *one-class* 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.

4.2 Feature extraction

In a preprocessing step, the large signals 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 the resulting feature datasets.

	$MFCC_1$	…	$MFCC_{13}$	MFD_1	…	MFD_{13}	*kurtosis*
Feature Vector 1	###	###	###	###	###	###	###
…	###	###	###	###	###	###	###
Feature Vector N	###	###	###	###	###	###	###

Table 1 Feature dataset schema

4.2.1 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 Fourier 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 condition signal.

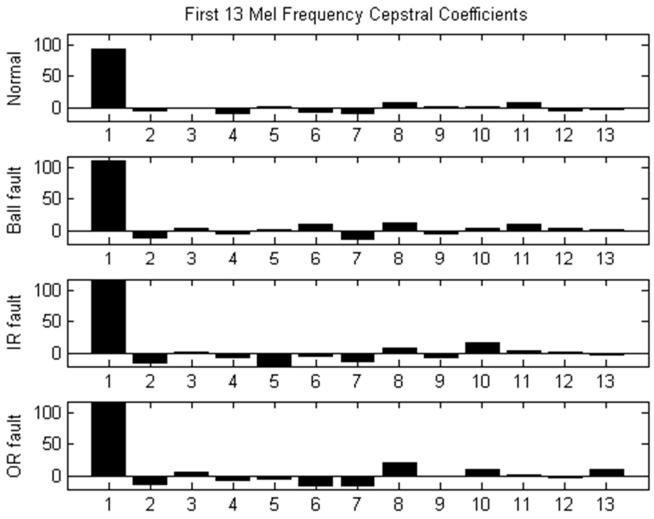


Fig. 15 Thirteen Mel Frequency Cepstral Coefficients

4.2.2 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 k_{max} , 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 *HFDs* where calculated by selecting a different value for k_{max} each time. With this approach, the best k_{max} values could later be selected indirectly by *PCA* or similar *feature selection* methods.

The graph in figure 16 illustrates *HFD* values of the first 15 *feature* vectors of each condition signal, with $k_{max} = 6$.

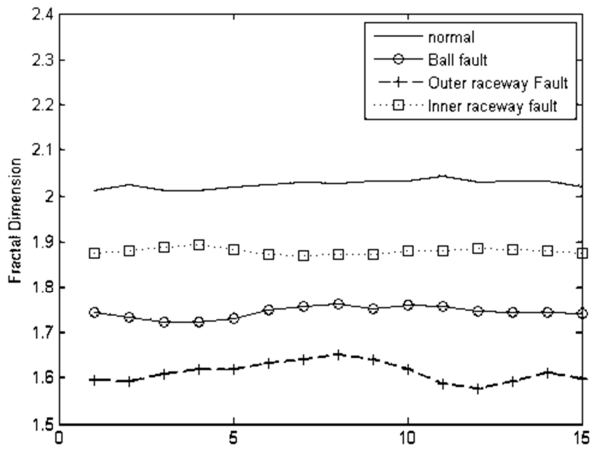


Fig. 16: 15 Higuchi Fractal Dimensions with k=6

The *fractal dimensions* of the normal signal obviously exceed the expected fractal dimensions of a curve, which should lie in the 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 the 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 datasets.

4.3 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.

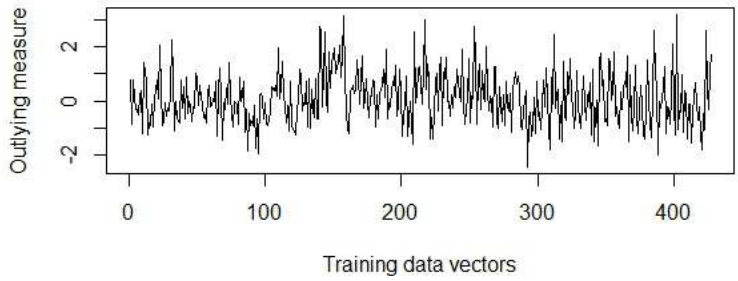


Fig. 17 Outlier measures of Roller Bearing normal training data

According to a rule of thumb given in (Breiman & Cutler, Random Forests), data points with an *outlier measure* above a threshold of about |10| require closer inspection. Figure 14 shows clearly, that the *outlier measures* for all training data points are smaller than |10|.

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

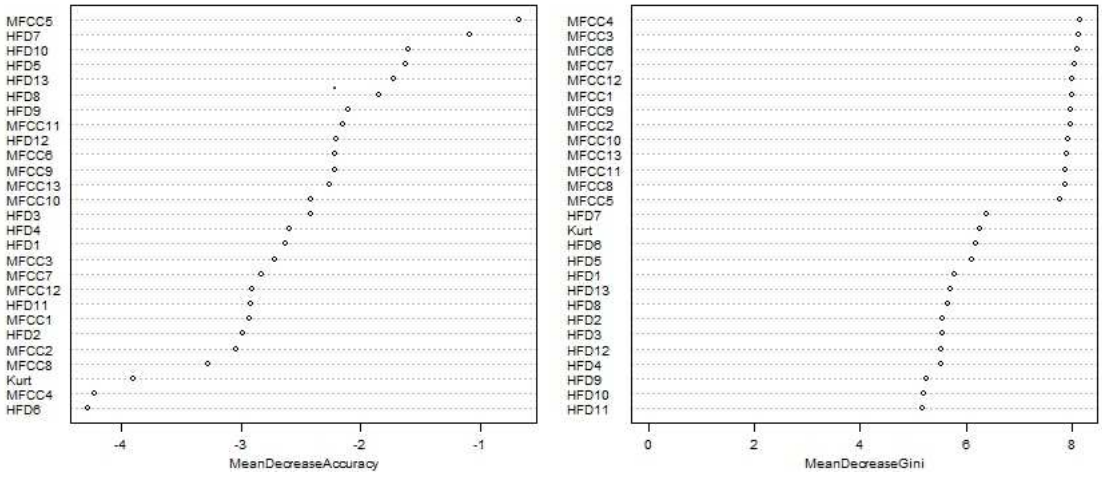


Fig. 18 Attribute Importance of Roller Bearing normal features training set

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 is evaluated in (4.4.2).

For the application of *random forests* in an *outlier detection* scenario, 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 (3.7.5), with a subset of the normal *feature* dataset
2. Calculation of a normal *class prototype* $\boldsymbol{p}_{\boldsymbol{normal}}$ (3.7.3)
3. Computation of the Euclidean distances $d(\boldsymbol{x}_n, \boldsymbol{p}_{\boldsymbol{normal}})$ between test objects $\boldsymbol{x}_n$ and the normal *class prototype* $\boldsymbol{p}_{\boldsymbol{normal}}$
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 *class prototype* and a test set consisting of 120 samples, as calculated during a test run of the classification approach described above.

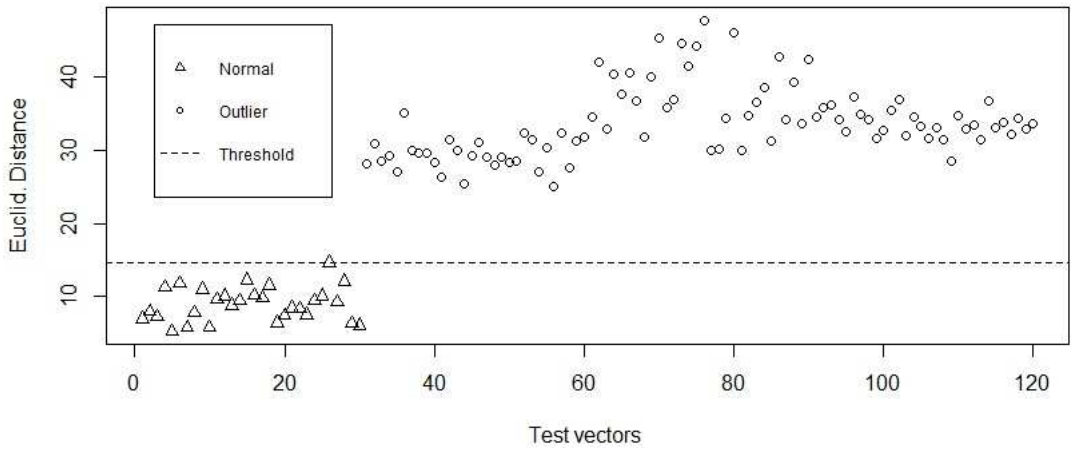


Fig. 19 Euclidean Distances between test data and normal prototype

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 *class prototype* (i.e. $F_{T-} = 0$), 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 $F_{T-} > 0$.

4.4 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 *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.

4.4.1 Performance evaluation

One experimental sequence comprised the following steps:

- Construction of a training set with 368 objects, sampled without replacement from the normal features dataset
- 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 dataset
- Training of each classifier with the training set
- Testing of each classifier using the test set
- Measuring of the training and testing runtime
- Calculation of the *error type 1* and *error type 2* for each classifier

The complete sequence was repeated ten times.

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

- training and testing time
- *type 1* and *type 2* errors
- $Accuracy = \frac{\#True\ Outliers + \#True\ Targets}{\#True\ Outliers + \#True\ Targets + \#False\ Outliers + \#False\ Targets}$

Table 4 contains a summary of the results of one complete experimental run, where F_{T-} was set to 0.1 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	$\#trees = 100$ $\#nodes = 737$ mtry=5	0.9432 sec	0.0057 sec	0.1044	0	0.9478
SV-dd	Gaussian kernel $sigma = 6$	20,2460 sec	0.0093 sec	0.5300	0	0.735
K-means	$errtol = 10^{-5}$ $k = 5$	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	$h = 1.5127$	0,2277 sec	0.0259 sec	1	0	0.5
SOM	$2D, \#neurons = 675$ $h = [0.6,0.2,0.01]$ $\eta = [0.5,0.3,0.1]$	162,045 sec	0.0058 sec	0.1733	0	0.9133

Table 2 Experimental results

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

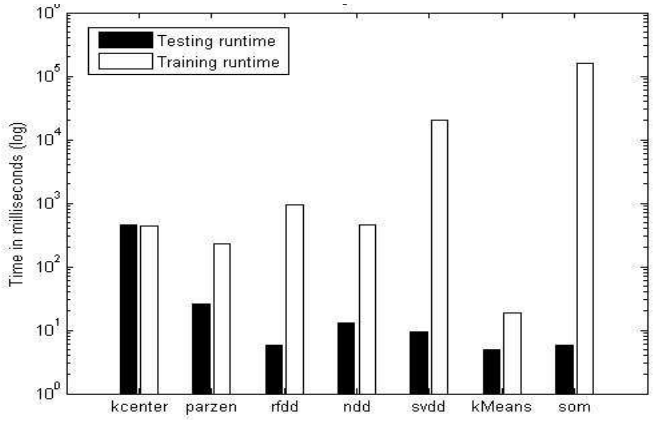


Fig. 20 Time Performance

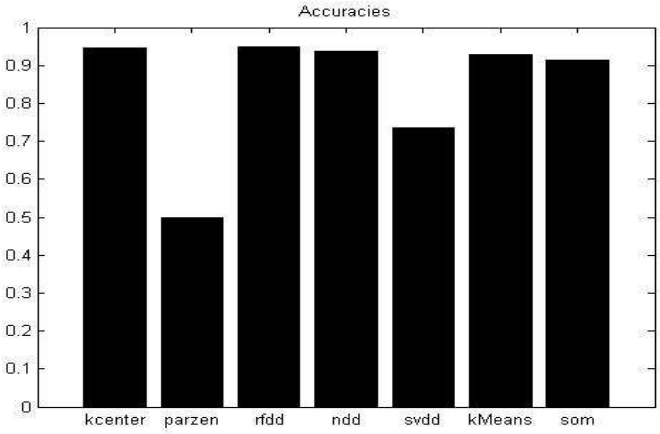


Fig. 21 Accuracies

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.

4.4.2 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 all *features*, and removing the least important feature in each iteration:
    - 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 dataset* are illustrated in figure 22.

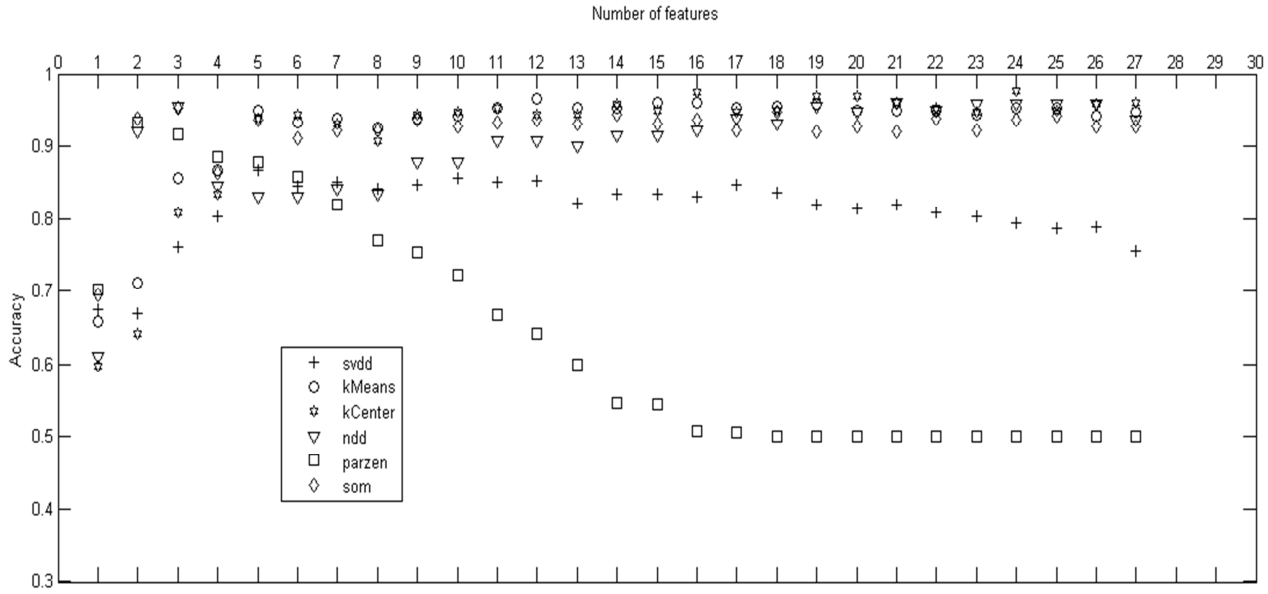


Fig. 22 Accuracies with reduced features

Figure 19 shows, that for most *classifiers* the number of *features* could be reduced considerably, without sacrificing *accuracy*. This could be an indication of correlations between individual *features*.

The Parzen window showed an opposite trend compared with most other classifiers. Starting from 16 *features*, its *accuracy* improved gradually until a maximum is reached at 3 *features*. 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 setting.

5 Conclusions

This Thesis presented a complete computerized *outlier detection condition monitoring* scenario, including data preprocessing, *feature extraction*, *feature reduction*, *one-class-classification* and evaluation. The combination of *feature* extraction and classification methods produced altogether good results for the *roller bearing* dataset.

The *random forest* based *one-class-classification* approach introduced in this Thesis proved to be highly successful applied to the *roller bearing* dataset. Furthermore it was shown, how some of the *random forest* features could be used to gain insight into dataset characteristics in a *one-class* problem. The general performance of the *RFDD* classifier could be improved by calculating more than one *target class prototype*, which would extend its applicability to more complicated *one-class-problems*. Research on methods to calculate multiple *class prototypes* based on *random forest* features could be part of a future Thesis.

A general problem in *outlier detection* scenarios of the type simulated in this Thesis is that nothing is known about the distribution of the *outlier* "class" at training time. A large *target* sample set combined with a tight boundary improves the performance of *one-class classifiers*. Reliability of the results could possibly be further increased, by training and using several different models in parallel and using a majority vote to classify new objects.

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 .

Index

German Summary

Die Analyse von Wälzlager-Vibrationssignalen ist ein Standardproblem in der Zustandsüberwachung von Maschinen, da viele Maschinendefekte mit Lagerfehlern zusammenhängen. (Nelvamondo, Marwala, & Mahola, 2006). Dementsprechend wurden Wälzlagersignale bereits in vielen Publikationen verwendet um Methoden der Zustandsüberwachung zu entwickeln und zu evaluieren (Marwala, 2012), (Nelvamondo, Marwala, & Mahola, 2006), (Li, Chow, Tipsuwan, & Hung, 2000). In all diesen Arbeiten wurden sowohl Signale des normalen Zustands als auch Signale von Fehlerzuständen verwendet um überwachte Entscheidungsalgorithmen wie Neuronale Netze oder Support Vektor Maschinen zu trainieren.

Ziel dieser Thesis ist die Präsentation eines beispielhaften Ausreißer-Detektion Ansatzes unter Verwendung eines bereits existierenden Wälzlager Datensatzes (Case Western University). Dabei werden im Unterschied zu den genannten Arbeiten ausschließlich Normalzustands-Signale eingesetzt um einen bestimmten Typ von Entscheidungsalgorithmen zu trainieren, die als „One-Class Classifier“ bekannt sind. Die Signale von Fehlerzuständen werden dann zur Evaluation dieser Entscheidungsalgorithmen eingesetzt.

Meine Arbeit umfasst die folgenden Aspekte:

- Recherche zum Thema Zustandsüberwachung und die Präsentation einer allgemeinen Vorgehensweise in der rechnergestützten Zustandsüberwachung (1.1)
- Recherche von Methoden zur Analyse von Zeitsignalen („Feature Extraction“) und Auswahl geeigneter Techniken im Hinblick auf die Wälzlagersignale (2,4.2)
- Implementierung einer Methode zur Berechnung von Fraktaldimensionen nach Higuchu (*Higuchi Fraktal (HFD)*) (4.2.2)
- Recherche zum Thema „One-Class Classification” und Auswahl geeigneter Methoden in Verbindung mit den Datensätzen und den Datenanalysemethoden (3,4)
- Design und Implenetierung einer neuen “One-Class Classification”-Methode, basierend auf Random Forest (3.7.6,4.3)
- Implementierung und Evalutation eines kompletten Ansatzes zur Ausreißer-Detektion, der Datenanalyse, Feature-Auswahl (feature selection), Klassifizierung von Daten und die Auswertung der Ergebnisse umfasst (4).