1. Introduction

Condition Monitoring is the systematic surveillance of industrial machines or structures with the aim to estimate and predict the health of the monitored system. In manufacturing, the monitoring and classification of machine states can help to prevent expensive production losses caused by unforeseen machine failure and reduce maintenance costs by allowing a more systematic approach to fault classification and lifetime prediction. 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 typical Condition Monitoring sequence inspired by a general theory of Condition Monitoring given in (Marwala, 2012).



Fig. 1 Condition Monitoring Framework

The several steps shown in figure 1 are summarized in the following paragraphs with an emphasis on the aspects relevant for this thesis.

* 1. Data Acquisition

The data acquisition step comprises the selection and setup of appropriate devices for data acquisition and the actual process of collecting data from a machine. Typical data acquisition devices used in machine monitoring are accelerometers, measuring vibrations at different parts of machines.

* 1. Data Analysis

The Outputs of the data acquisition system of the Machine Condition Monitoring scenarios considered in this thesis, can be represented by a sequence 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 a time . This kind of raw often represents a sampled vibration signal and can be analyzed with certain statistical measures such as variance, kurtosis or skewness. However, many machine learning techniques used in the classification of machine conditions perform poorly with time signal data and need preliminary data processing step which derives meaningful *Features* from the raw time data.

A number of techniques can be applied to identify and quantify aspects of the initial raw time series data which can be used by *Classification Techniques* to distinguish machine conditions. Widely used are transformations techniques such as *Fourier-* or *Wavelet Transform*, other methods include *Cepstral* or *Fractal* analysis which are applied in this thesis.

The outputs of the *Data Analysis* step usually are data vectors of the form

where *I* denotes a specific data vector in a sequence ,…, are the particular *Feature* values of that specific data vector and *D* is the dimension of the *Feature Set*.

* 1. Feature Selection Techniques

The number of features can have a significant effect on the performance of classification algorithms, of which many are susceptible to the *Curse Of Dimensionality*. i.e. show an exponential increase in complexity with the number of features. Different *Feature Reduction* techniques try to tackle that problem by selecting the features which are best suited to distinguish machine conditions and discarding the others. By far the most popular of these methods is the *Principal Component* analysis.

* 1. Classification Techniques

A huge variety of methods exists that can be used to estimate machine condition from machine data. The available methods include statistical approaches, Machine Learning and Pattern Recognition techniques or mixtures of several techniques, also known as *Batch* or *Boosting* algorithms (Marwala, 2012). Many techniques involve an offline training phase, in which sample data is used to train a model that can be used to classify new data during machine running time.

The selection of decision techniques for a particular problem among other things strongly depends on the data available in the particular scenario. One important aspect in this regard is the question if sample data is available which can be used to train a model.

*Supervised* techniques require data samples representing all possible machine conditions to train a model, which can be used to estimate all possible machine conditions from new data. Most of the popular *Machine Learning* methods such as Artificial Neural Networks or Support Vectors belong to this category.

If representative sample data is available for only one condition, *semi-supervised* techniques can be used to train models that are usually only able to distinguish one machine condition from all other possible conditions. *Semi-supervised* scenarios typically occur if it’s either too expensive or even impossible to obtain sample data covering all possible conditions. In *Machine Condition Monitoring* data representing normal machine conditions can often be obtained rather 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 Detection* (Banerjee, Chandola, & Kumar, 2007).

*Semi-supervised methods* are often modified versions of *supervised* techniques and are usually referred to as *One-Class-Classifiers* (Tax M. J., 2001).

If no sample data is available, *unsupervised methods* can be used to detect patterns in datasets. *Unsupervised* techniques involve genuine *Pattern Recognition Methods* such as *k-Nearest-Neighbor,* but also supervised techniques which can be run in an unsupervised mode.

* 1. Outlook of this thesis

Goal of this Thesis is the realization and evaluation of an exemplary semi-supervised *Machine Condition Monitoring* scenario. For this purpose, a benchmark dataset is selected and analyzed by a combination of *Feature Extraction* and classification techniques.

As benchmark dataset, the *Roller Bearing Dataset* (see 2…)was selected which is available from a website of the *Case Western University* (Case Western University)and has already been used in several publications (Marwala, 2012), (Nelvamondo, Marwala, & Mahola, 2006). While in most research papers both the normal and fault condition samples were used to train supervised classifiers, in this thesis only the normal condition samples are used for training. The available fault condition samples are then utilized as an optimal test set for the evaluation of the feature selection and classification techniques.

The *Feature Extraction* techniques applied in this thesis are the *Cepstral Coefficient Analysis*, *Multi Fractal Dimension* analysis and the *kurtosis* measure. All of these techniques have been successfully used in research (Marwala, 2012) (Nelvamondo, Marwala, & Mahola, 2006) and are briefly introduced in 2….

Several *semi-supervised* algorithms are used to detect faulty conditions in the machine data. A general introduction to the *One-Class-Classification* approach is given in … and several *One-Class-Classifiers* are briefly described in the following paragraph. The chapter concludes with a new approach for the application of *Random Forest* classifiers in a semi-supervised scenario.

Chapter 3 eventually explains the experimental setting and some implementation details, and provides a discussion of the results.

1. 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.

* 1. 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, Zustandsüberwachung von Maschinen , 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

* 1. 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
   1. 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).

1. 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 .

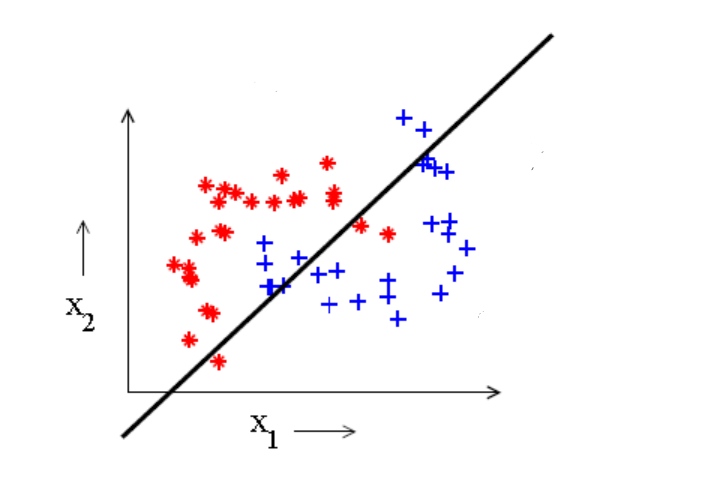


Fig. 2: Two-Class-Classification

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 C. M., 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.

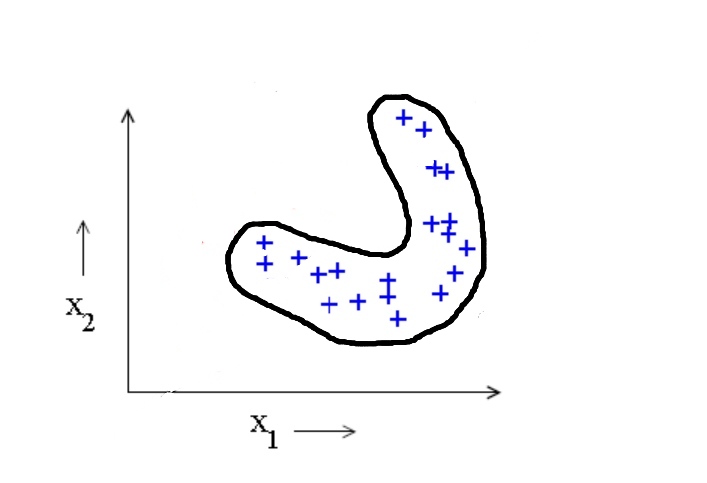


Fig. 3: One-Class-Classifier

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 number 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.

* 1. 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 C. M., 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 C. M., 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).

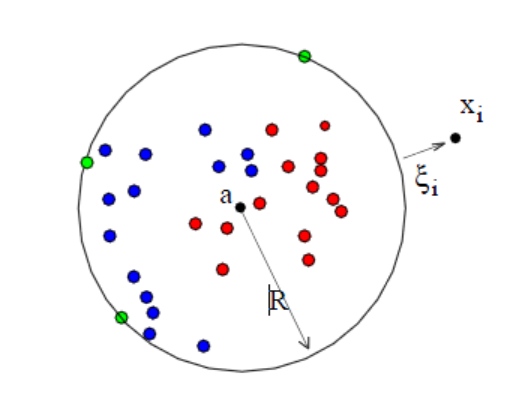


Fig. 5: Support Vector Data Description

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

* 1. 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

* 1. 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:

* 1. Nearest Neighbor Data Description

*K-Nearest-Neighbor* classification is based on a local density estimation of the form (Bishop C. M., 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

* 1. Parzen Window Data Description

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

* 1. 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.

1. Random Forests

*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. The features used in this thesis to analyze the data sets and to construct a *Random Forest* based *One-Class-Classifier* are introduced in the following chapters.

* 1. *Out Of Bag* error estimate

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.
   1. Variable Importance

A measure for the *Feature* 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.

* 1. Proximity

A *Proximity Matrix* in its basic form 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.

* 1. Prototypes

For each class c, 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.
   1. Unsupervised Learning

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*.

* 1. Outlier Detection

*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 .

1. Experiments

Figure… Provides an overview of the steps involved in the classification approach of this thesis.

The experiments in this thesis are based on the roller bearing data sets available at (Case Western University). A brief overview of the datasets is given in 4.1. The preprocessing and feature extraction steps are summarized in 4.2 and chapter 4.3 describes the classification process and results.

The complete classification approach – including preprocessing and feature extraction – was implemented in a Matlab script. 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 outlined in 4.3 was evaluated based on the popular R-package (Liaw & Wiener, 2002). For a seamless integration of the *Random Forest* *data description* method, a *Random Forest One-Class Classifier* was implemented based on a MATLAB adaption of the R-package (Jaiantilal, 2010). This *Random Forest One Class Classifier* was finally wrapped into a *Dd\_tools* mapping*.*

* 1. 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 research has been done to develop techniques for the classification and prediction of 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 with a set of rolling elements running between their tracks, as illustrated in Figure 12.

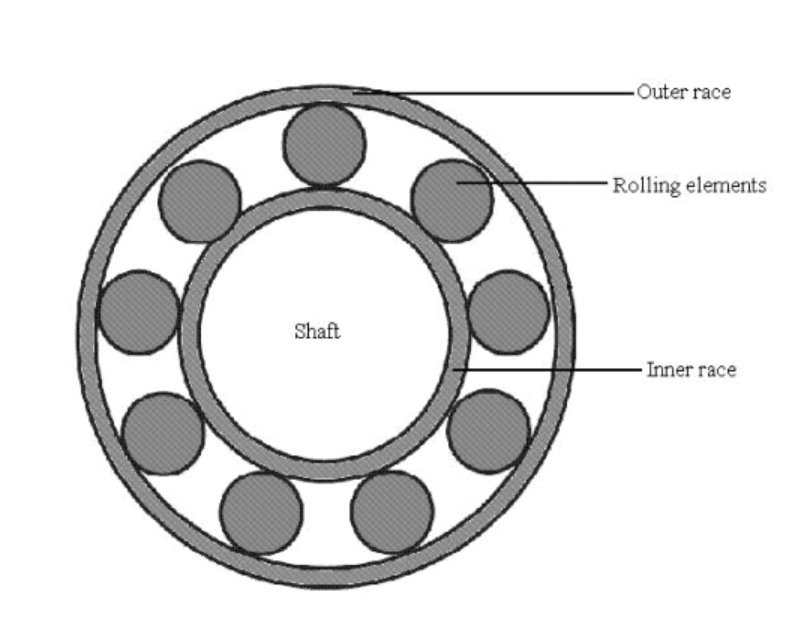


Fig. 9 Roller Bearing

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

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

A Roller Bearing Benchmark Dataset mentioned in several research papers is available at (Case Western University). The data collection contains vibration signals representing the following *roller bearing* conditions:

* Normal condition
* Ball fault condition
* Inner raceway fault condition
* Outer raceway fault condition

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 only four datasets representing the four main conditions where used, acquired from the drive end, at a speed of 1797rpm and with a sample rate of 48k. Figure 3 displays the first five revolution of each time signal.

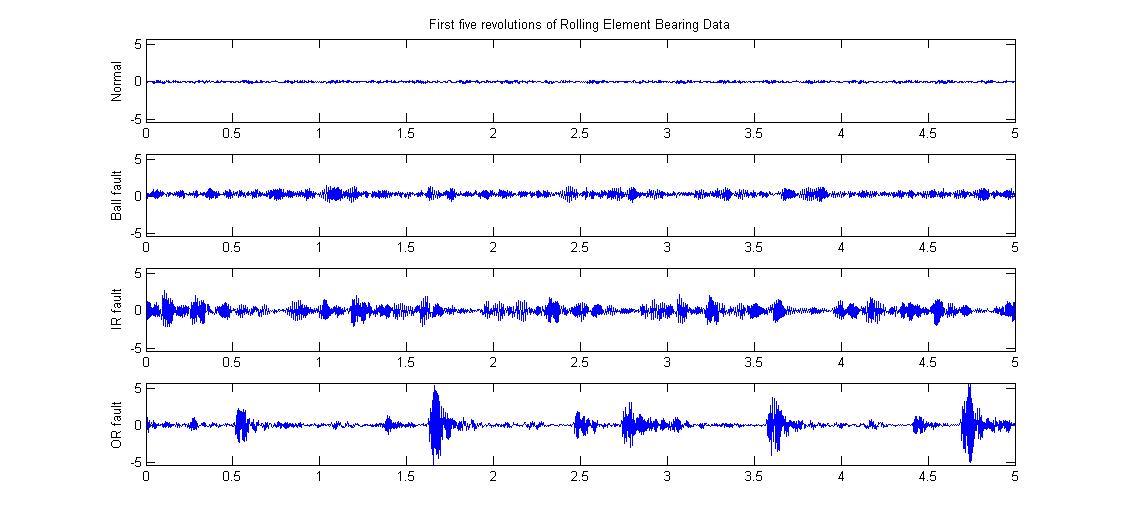


Fig. 10 Roller Bearing Sample Data

According to the *semi-supervised* scenario assumed in this thesis, only the normal signal was used to train *one-class classifiers.*

* 1. Feature Extraction

In a preprocessing step, the large datasets representing the normal and the three fault conditions were split into segments with a length of 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 the feature sets.

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

Table Feature data set schema

* + 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 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.

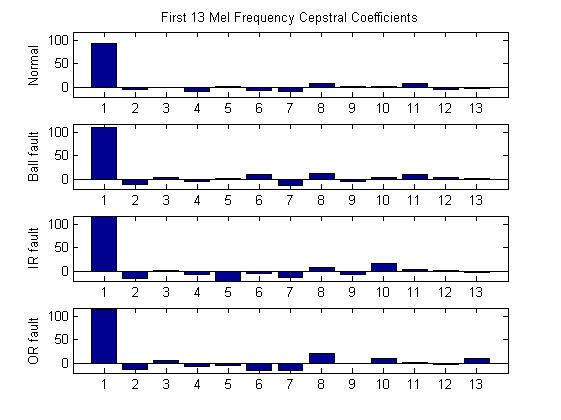


Fig. 11 Thirteen Mel Frequency Cepstral Coefficients

* + 1. Higuchi Fractal Dimensions

For *HFDs* feature extraction, a simple MATLAB implementation was used (Selvam, 2011) with just one free parameter (2.2). 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 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 can 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 .

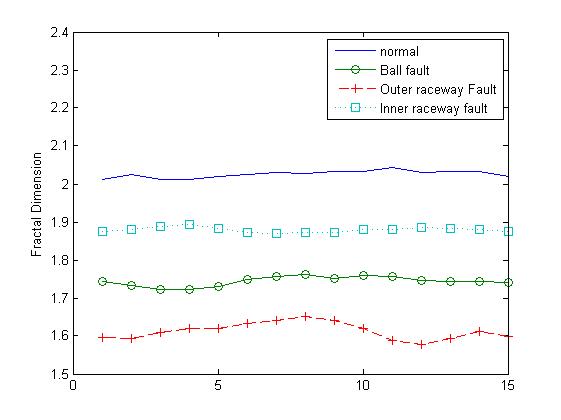


Fig. 12: 15 Higuchi Fractal Dimensions with k=6

Figure 2 shows, that *fractal dimensions* of the normal signal exceed the expected fractal dimension of a curve, which should lie in a range between one and two. The reason for this may be related to the specific shape of the normal signal as illustrated in figure 10, 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 suited to distinguish conditions. The graph of the first 15 *HFDs* suggests that *HFDs* are in this regard a good choice for extracting features from the given data sets.

* 1. 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. 12 shows 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 12 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 13 shows the *Attribute Importance Measures* calculated during a *Random Forest* training with 427 roller bearing normal 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. The *Attribute Importance* results were used later, to retrain the *Random Forest* using only the most important attributes.

For the application of *Random Forest* in *outlier detection*, a *random forest* *one-class classifier* was implemented according to the generic approach introduced in 3. The classification sequence consists of the following steps:

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

A prototypecalculated based on 382 normal samples is shown in table 1.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **MFCC1** | **MFCC2** | **MFCC3** | **MFCC4** | **MFCC5** | **MFCC6** | **MFCC7** | **MFCC8** | **MFCC9** | **MFCC10** | **MFCC11** | **MFCC12** | **MFCC13** |
| 93.814 | -4.799 | 0.910 | -7.189 | -1.213 | -4.746 | -6.296 | 6.458 | -1.015 | 1.593 | 4.271 | -1.854 | -5.961 |
| **HFD1** | **HFD2** | **HFD3** | **HFD4** | **HFD5** | **HFD6** | **HFD7** | **HFD8** | **HFD9** | **HFD10** | **HFD11** | **HFD12** | **HFD13** |
| 1.205 | 1.264 | 1.323 | 1.389 | 1.463 | 1.551 | 1.663 | 1.781 | 1.865 | 1.893 | 1.888 | 1.871 | 1.847 |
| **Kurt** |
| 2.773 |

Table Normal Prototype

Figure 11 illustrates the Euclidean distances between the normal class prototype and a test set consisting of 120 samples from the normal, the ball fault, the Inner Raceway fault and the Outer Raceway fault sets.



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. ), clearly separates normal data from outlier data in this scenario. For robustness against outliers, the threshold should be calculated with . Table 2

Table 2 shows the *confusion matrix* for this classification run.

|  |  |  |
| --- | --- | --- |
|  | Target  (Predicted) | Outlier  (Predicted) |
| Target  (Actual) |  |  |
| Outlier  (Actual) |  |  |

Table 3 Random Forest One Clas Classifier Confusion Matrix

* 1. Classification

The *One-Class-Classifiers* were trained with a randomly sampled subset of the normal features datasets. The trained classifiers were then tested with a test set sampled equally from the set of normal data not used during training, and the data sets representing fault features. This training and testing sequence was repeated several times, each time with a different sampled training and test set. To evaluate the classifiers, the errors were finally averaged over all training and test runs.

1. Conclusions

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **D**  **Sensitivity** | **Parameters** | **Training**  **Costs** | **Testing**  **Costs** | **Storage**  **Costs** | **Parallel-**  **lization** |  | ***Acczracy*** |
| ***SVDD*** |  | k | constant |  | Constant  (k support objects) |  |  |  |
| ***K.Center*** |  |  |  |  |  |  |  |  |
| ***K.Means*** |  | none |  |  |  |  |  |  |
| ***NN*** |  |  |  |  |  |  |  |  |
| ***Parzen***  ***Window*** |  |  |  |  |  |  |  |  |
| ***SOM*** |  |  |  |  |  |  |  |  |
| ***Random***  ***Forest*** |  |  |  |  |  |  |  |  |