Es wurden keine Einträge für das Inhaltsverzeichnis gefunden.

Semi-Supervised Methods in Condition Monitoring

1. Condition Monitoring

Condition Monitoring refers to the systematic surveillance of industrial machinery or structures with the aim to detect faults and damages in an early stadium. In manufacturing, the monitoring and classification of machine states can help to prevent expensive production losses due to unforeseen downtimes of machines and reduces maintenance costs by supporting a more systematical approach to fault classification and lifetime prediction. In other areas such as aerospace or construction, Condition Monitoring methods are used to prevent or predict faults in structures which can result in severe accidents.

In the past, Condition Monitoring in manufacturing environments was usually conducted by experts with years of experience. This approach implied the personalization of essential knowledge, combined with the relatively high costs for experts and was generally a bad use of resources. With the spread of computational methods in manufacturing, more and more steps in Condition Monitoring processes where conducted by computers, but still experts where needed to analyze and draw conclusions from data collected by computers. Within the past 20 years, sophisticated methods from the areas of statistical computing, machine learning and pattern recognition where developed to support or replace human experts in Condition Monitoring (“Expert Systems”?)

Figure 1 illustrates a generalized theory of Condition Monitoring, described in (Marwala, 2012).

The data acquisition device comprises devices and infrastructure used to obtain useful data from processes or systems. This is usually done by sensors such as accelerometers, which are connected to computers.

The Data Analysis Device involves techniques concerning the representation of data. Raw data collected by sensors is represented in the time domain with discrete sensor values at certain points in time. Depending on the domain and the methods used in the following step, time domain data is not meaningful enough or completely useless. In such cases, techniques can be used to transform time domain data into the Frequency Domain (Fourier Analysis), the Time-Frequency Domain (Short Time Fourier Transform, Wavelet Transform) or other domains.

Feature Selection as a third step is used to identify and select aspects of the collected data that are best suited to classify and predict fault states. Feature Selection techniques used in Condition Monitoring involve general methods such as Principal Component Analysis but also more specialized techniques such as Mel Frequency Cepstral Coefficients or Fractal Analysis.

The Decision Making Device can be any – combination of - Statistical, Machine Learning, or Pattern Recognition algorithms which takes condition monitoring data as an input and produce state classifications as output. Popular techniques used in Condition Monitoring are Parametric and Non-parametric statistical Models, Artificial Neural Networks, Support Vector Machines and Hidden Markov Models.

The final step in a generalized Condition Monitoring scenario is the diagnosis and prediction of system states, based on the results of the decision step.



Fig. 1 Condition Monitoring Framework

1. Condition Monitoring Data

Condition Monitoring techniques usually process time series data, with datasets representing some kind of sensor values at a certain point in time. In structural analysis problems, data can be obtained by exiting the structure and measuring the response. In the Condition Monitoring of Machines, vibrational data generated during running time of machines is used to

Another categorization of Data in a general theory of Computational Intelligence Methods is based on the availability of classified training data, which determines the choice of decision algorithms in phase 4. In a *supervised* setting, training data with samples for all classes to be predicted can be used to train a classifier. If the training data contains only samples of one class, *semi-supervised* decision techniques can be trained to detect all instances of the known class and reject anything else as outlier data. If no training data is available, *unsupervised* techniques have to be used which typically include a combination of various statistical measures.

The *semi-supervised* scenario is very common in Condition Monitoring, since it is often much easier and cheaper to obtain data from a system under normal condition than from a faulty system. In some domains, such as aerospace, it is not even possible to obtain a representative amount of fault data which could only be generated by accidents or catastrophes.

In a *semi-supervised* Condition Monitoring Setting, usually data representing the normal state of a system is available as training data. This data can be used to train classifiers that accept instances pertaining to the normal class and reject any other data instances as outliers.

In this thesis, *semi-supervised* techniques used for predictive maintenance are discussed and a *semi-supervised* variation of Support Vector Machines is implemented and tested. The data used to train and test the algorithm is described in the next section.

1. Rolling Element Bearing Data

Rotating Machines are very common in various industrial applications, but also in many other areas such as Aerospace or Power Generation. 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).

A popular Rolling Bearing Benchmark Dataset mentioned in many research papers can be found at (Case Western University). The data was obtained through accelerometers attached to a rotating machine with a central Rolling Bearing Element. Rolling bearing consist of two concentric rings, the Inner and Outer Raceway with a set of rolling elements running between their tracks as illustrated in Figure 2



Fig. 2 Rolling Element Bearing

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

Rolling Bearings generate vibration signals with characteristic signatures, according to the states of the raceways and the rolling elements. The Rolling Bearing Dataset from the Case Western University contains time-series accelerometer data obtained from different positions of the machine under normal condition and under several fault conditions. Figure 3 shows sample data of the Rolling Bearing under normal condtition, with Ball Faults and faults of the Inner and Outer Raceway.



Fig. 3 RollingBearing Sample Data

In this thesis, an artificial semi-supervised scenario is created by using only the normal data to train a classifier which accepts normal data and rejects any other data as outliers. The available fault-state data of the Rolling Bearing Elements is then used to test and evaluate the model.

1. Feature Extraction

Datasets occurring in the Condition Monitoring domain are often rather large time-series datasets which are expensive to process directly. Besides, data aspects linked to normal or faulty system states are in many cases hidden in the time series data and cannot be detected directly. Successful Condition Monitoring Frameworks therefore usually combine some kind of preprocessing and Feature Extraction techniques with a suitable decision algorithm.

“The robustness of a classification system depends on the usefulness of the extracted features and the reliability and effectiveness of a condition monitoring classification system”.

Several techniques have been applied to extract useful features from the Rolling Bearing dataset (Marwala, 2012). Some successful approaches combined Mel Frequency Cepstral Coefficients (MFCC), Multi Fractal Dimensions (MFD) and Kurtosis measure as feature extraction techniques with machine learning algorithms such as Neural Networks, Hidden Markov Models and Support Vector Machines to classify faults (Marwala, 2012), (Nelvamondo, Marwala, & Mahola, 2006). MFD, MFCC and Kurtosis are applied in this thesis for their proven ability to extract meaningful features from the Rolling Element Dataset time series. The techniques are introduced in the following section.

* 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

(4.1.1)

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 which is characterized by its impulse response . The output is given by the convolution of the input signal and the impulse response

(4.1.2)

In the Frequency domain, the convolution in (4.1.2) is transformed into the multiplication

(4.1.3)

With (4.1.1) and (4.1.3) the cepstrum of an *LTI* is

(4.1.4)

From (4.1.1) and (4.1.4) 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, the 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 as Features 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 frequency domain by applying the Fast Fourier Transform:

Where is the number of frames, and is the Hamming Window given by

with and a normalization factor .

1. Changing the frequency spectrum to the Mel scale with 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 that consist 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 in Condition Monitoring for the extraction of features from machine-vibration data (Nelvamondo, Marwala, & Mahola, 2006), (Marwala, 2012). In this thesis, Higuchi’s method is used for its efficiency (simplicity?) and its proven reliability in various applications.

Given a sampled 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 *HFD* as the slope of the least squares linear best fit line for the curve versus , since
   1. Kurtosis

Generally, Kurtosis is any measure for the Peakedness of a curve. 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).

In a preprocessing step, the large datasets representing the normal and the different fault states were subdivided into segments of equal length, with data from five revolutions of the Rolling Bearing per segment. The Kurtosis was calculated directly from each segment. For the extraction of MFD and MFCC, each segment was further divided into 14 frames of equal length. Figure 3 shows the MFD features of 14 sequences:



Fig. 4 MFD features of one data segment

Figure 4 illustrates the MFCC features of one sequence:



Fig. 5 MFCC features of one sequence

1. Classification
   1. Support Vector Machines

A linearly separable two-class dataset in n-dimensions can be separated by an (n-1)-dimensional hyperplane of the form

(5.1.1.)

as illustrated in figure 4 for a 2-dimensional dataset with two classes.

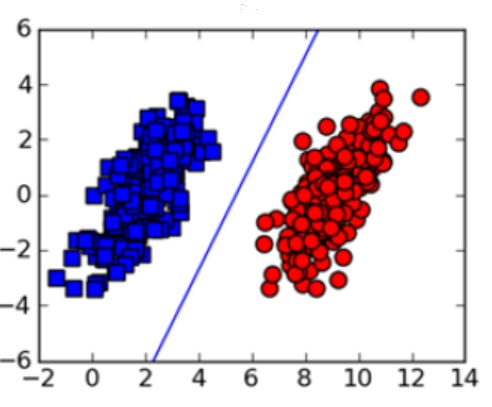


Fig. 6 Linearly separable two class dataset

For a dataset with data vectors  and the corresponding class labels where , at least one choice of the parameters and b can be found such that 5.1.1. satisfies

and

.

A support vector machine finds optimal values for the parameters and b by maximizing the margin, which is defined as the perpendicular distance between the hyperplane and the points closest to the hyperplane, as shown in figure 5.

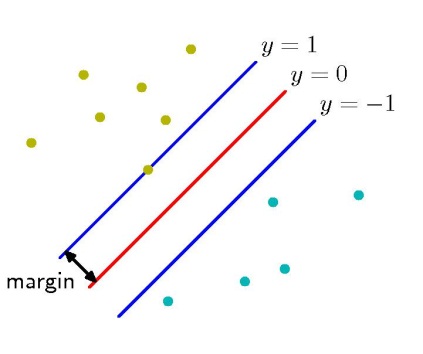


Fig. 7 The concept of margin

The solution for this optimization problem and the subsequent classification of test points solely depend on the points closest to the hyperplane marked by a circle in figure 6, which are known as the support vectors because they define and support the decision boundary.



Fig. 8 The Support Vectors

The perpendicular distance of a point to a hyperplane of the form 5.1.1. is given by

With for all correctly classified points of a training set, the distance of a training vector to the decision surface is given by

(5.1.2)

The support vector approach tries to find the closest points to a hyperplane and then adjusts the parameters andb such that the distance (5.1.2) between the closest points and the hyperplane is maximized. This maximum margin solution can be found by solving (Bishop C. M., 2009)

(5.1.3)

If (5.1.1) is set to

for the points closest to the hyperplane then the constraint

(5.1.4)

holds for all training data points and

reducing the maximization problem in (5.1.3) to the maximization of , which is equivalent to minimizing

(5.1.5)

subject to the constraints given by (5.1.4). With the introduction of Lagrange Multipliers for the constraints defined by (5.1.4), this optimization problem can be transformed into the Lagrange function

(5.1.6)

Setting the partial derivatives of (5.1.6) with respect to and to zero results in the conditions

(5.1.7)

and

which can be plugged into (5.1.6) to give

(5.1.8)

The last expression is also known as the dual representation of the optimization problems. The dot product of the training vectors in (5.1.7) can now be replaced by a kernel , which implicitely transforms the training data into a potentially infinite feature space and modifies (5.1.7) to

(5.1.9)

The introduction of a kernel makes the classifier considerably more flexible by allowing the substitution of a variety of valid kernels and extends the application of support vector machine to non-linear datasets through the implicit feature space transformation.

The maximum margin problem can now be solved by maximizing (5.1.8) with respect to the constraints

(5.1.10)

If hte expression for derived in (5.1.7) is plugged into the initial hyperplane formula (5.1.1) , new test points can now be classified by evaluating the sign of

(5.1.11)

Where is only true for the training points closest to the hyperplane and the corresponding are known as the support vectors, since the classification exclusively depends on them.

The optimization problem (5.1.9) with the constraints given by (5.1.10) is based on the assumption that data is linearly separable in the feature space to which it is implicitly transformed to by the kernel function. In case of overlapping data distribution in feature space this approach can result in a high error on the training data and in a classifier with poor generalization. This can be prevented by relaxing the hard constraints given by (5.1.4) through the introduction of slack variables such that each training data point satisfies

(5.1.12)

with for data points that are on or within the correct margin boundary and for any other training data point, as illustrated in figure … for the case of 2 dimensional data.

The maximum margin can now be found by minimizing

(5.1.13)

Where C regulates the tradeoff between the influence of the slack variables and the margin and thus controls the overall error.

With the introduction of and the corresponding constraints for with N being the number of training points, the Lagrangian (5.1.6) now extends to

(5.1.14)

Setting the partial derivatives with respect to ,b and to zero and using the results to eliminate ,b and from (5.1.13), the dual representation of the margin maximization problem again results in

Which is identical to the separable case (5.1.9) but with different constraints, which are now

(5.1.15)

The standard Support Vector Machine as introduced above is a supervised classification algorithm, which is in its basic form able to separate two classes and can be extended to multiclass and regression problems.

An approach to apply Support Vector Machines to semi-supervised or outlier detection problems was introduced by Tax and Duin as *Support Vector Data Description* (SVDD) (Tax & Duin, 2004). Instead of using a hyperplane to separate two or more classes, the SVDD tries to surround the available training data by a minimal hypersphere, defined by its radius R and the center **a**. The data enclosed by that hypersphere is the target class, data outside boundary is considered outlier data.

Similar to the optimization problem (5.1.5) in the training of a supervised Support Vector Machine, the intention is to minimize

subject to the constraints

(5.1.16)

Where and R are the center and the radius of the hyperspere and is the n-th training data vector, with the assumption that the training data does not contain outliers.

To allow the occurrence of outliers data in the dataset, the constraints in (5.1.13) have to be softened to a certain degree by the introduction of slack variable, such that the softened constraints are given by

(5.1.17)

and the decision boundary changes to

(5.1.18)

where . Similar to (5.1.13), C controls the tradeoff between a tight decision boundary and the influence of outliers.

A compact formulation of the optimization problem is again obtained by combining the objective function (5.1.17) and the corresponding constraints (5.1.16) to a Lagrange Multiplier, which is given by

(5.1.19)

To minimize the Lagrangian, its partial derivatives with respect to and must be set to zero, which results in the new set of constraints

(5.1.20)

(5.1.21)

Where , is required for all Lagrange multipliers and because of 5.1.18, and thus (5.1.21) implies

(5.1.22)

Substituting the relations in (5.1.19) into the Lagrangian (5.1.18) results in the dual representation of the SVDD problem, which is given by

subject to the constraints (5.1.21), where the inner products can again be replaced by kernels and the corresponding to the Lagrange multipliers for which are the *Support Vectors.*

Substituting (5.1.20) for **a** in the initial hypersphere constraint (5.1.16), a test object **z** is accepted if it satisfies

(5.1.23)

and rejected as an outlier otherwise.

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. 9 SVDD with RBF, sigma=2



Fig. 10, SVDD with RBF, sigma=3



Fig. 11, SVDD with RBF, sigma=5

* 1. Gauss Data Description
  2. K-Center based Data Description

The *Domain Approximation* algorithm introduced in (Ypma & Duin, 1998), a subset of *k* support objects is selected from the *N* samples of a dataset *X*. Each of the support objects is surrounded by a *receptive field* of the same 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 subset with the minimal radius r such that all data samples in ***X*** are covered by the receptive field of a 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

After training of the *k* support objects, the distance of a test object ***t*** 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 the data objects and their prototypes. The main differences to *k-centers* are the placement of the cluster centers or 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).

**Training**

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.

**Classification**

One-Class-Classification in *k-Means* can be done by defining a distance threshold and then using the classification function

to classify a test object as target or outlier. I.e. a test object is accepted as belonging to the target class if its distance to the nearest prototype is smaller than the threshold value and rejected as outlier otherwise.

* 1. PCA Data Description
  2. K-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 introduced by Tax (Tax M. J., 2001), a test object is accepted when its local density is larger than or equal to the local density of its *k* neighbors. In the case of the corresponding classification function is given by

Where is the nearest neighbor of in the training set and is the indicator function which is 1 if the inequality in … is true, and 0 otherwise.

* 1. Parzen Window Data Description

*Parzen Window* is a density estimation method where a kernel is centered on each of the training data objects and the density at a test point is estimated by accumulating the kernel functions with input over all training data points. 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.

* 1. Random Forests

*Random Forests™* (Breiman , Random Forests, 2001)are an ensemble learning method for classification and regression which grows several trees, in which the construction of each tree is based on a sampled subset of the complete training data. Classification is done by simply taking the majority vote among all trees as the class of a test vector.

The Training phase of a random forest involves the following steps for each tree:

1. If the number of cases in the original training set is N, N cases are sampled from the original dataset, with replacement.
2. If the number of attributes in the training set is M, a number m<<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 split the node.
3. Each tree is grown 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 (Breiman , Random Forests, 2001).

*CART* is a decisions tree algorithm which uses the gini criterion to split nodes and only allows binary splits. This restriction in many cases simplifies the splitting decisions, but may result in trees that are hard to interpret (Quinlan & Kohavi , 1999).

The split decision is made by selecting at each node the attribute resulting in the highest information gain, given by

(5.2.1)

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

(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 (Breiman & Cutler). 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

The first three features are a result of the basic tree growing process, while other features rely on optional computations and data structures which can easily be integrated into the tree construction. Some of these computations are introduced in the following subsections.

* + 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. Thus, about of the training cases are not used for the training of the nth 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:

* Each case in the original training set is put down the trees in the forest, in which it was not used in constructing the tree *(oob)*
* The number of cases for which the majority vote of the trees did not correspond to the actual class divided by the complete number of cases is the generalization error estimate for the random forest
  + 1. Variable Importance

A measure for the attribute importance can be obtained by randomly permuting the values of an attribute in a dataset as follows:

* Each case of the original training set 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 *c*
* The values of one attribute *m* in the dataset is randomly permuted
* The first step is repeated with the permuted dataset and the number correct votes is counted by *p*

An estimate for the importance of attribute *m* is then given by

Where is the number of trees in the forest.

* + 1. Proximity

A *Proximity Matrix* in its basic form is an NxN matrix, where N is the number of training cases used in building the forest and each entry of is a measure for the similarity or proximity between cases and . The matrix entries are computed during the forest construction as follows:

* After a tree is built, the complete original training set is put down the tree
* If cases and end up in the same leaf, their proximity is incremented by one
* After the complete forest is built, the proximities are normalized by the number of trees

From its definition it’s easy to see that the proximity matrix is a diagonal matrix with entries bounded above by one.

Several other useful features of a *random forest* are based on the proximity matrix, such as the evaluation of *Prototypes,* clustering and the detection of outliers, described in 5.2.5.

* + 1. Prototypes

Prototypes give an insight into the relation between the classes and the attributes in a dataset. They are found using the *Proximity Matrix* by

* For each class *c* finding the case with the most other cases of class *c* among its *k* nearest neighbours.
* Calculating the median, the 25 percentile and the 75 percentile for each attribute among these *k* cases
* Construction of an artificial new case consisting of the calculated medians, which represents the prototype of class *c*.
  + 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 vector is created by uniformly sampling from the value distribution for each attribute. This procedure is repeated *N* times, where *N* is the number of data vectors in the original dataset. The original training data and the synthetic data can then be combined and used to construct a two-class random forest with all of the original options such as *Outlier Detection*, *Variable Importance* measurement and missing value replacement. One of the important applications of the results of an unsupervised random forest is clustering, based on a dissimilarity measure derived from the *proximity matrix* (Shi & Horvath, 2006)*.*

* + 1. Outlier Detection

Based on the *proximity* measure, outliers can be defined as data vectors whose *proximity* to all other data is generally small. *Random Forest* defines outliers with respect to their classes as points in class whose proximity to all other points in is small. The average proximity of a point *x* to all the data vectors in class *c* is given by

A raw outlier measure for point with respect to class *j* can then be 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 points in class and is the *Median Absolute Deviation* of all raw outlier measures in class .

* + 1. Scaling

*Multidimensional Scaling (MDS)* is a generic term for several related statistical techniques for exploring similarities or dissimilarities in data. *MDS* techniques are often used to transform internal relations of high dimensional data into lower dimensions for visualization purposes.

For a collection of items and a distance measure between items and , a dissimilarity matrix can be defined as

Based on , *MDS* techniques try to find vectors such that

Where is a *vector norm*. Thus, *MDS* embeds the initial data items into a space such that the original distances between the items are preserved. If or , the similarity between data items can be visualized.

* 1. Roller Bearing Outlier Detection with random forests

Based on the *semisupervised* Condition Monitoring scenario introduced in 1.3, only features representing normal conditions of the roller bearing were used to construct a *Random Forest*. For evaluation of the *Random Forest* classifier, both normal and fault condition data was used.

In unsupervised training mode of Random Forests, all available training data is considered to belong to a single class. Based on the training data, a synthetic dataset representing a second class is created, according to the algorithm described in 5.2.5. This balanced dataset is then used to construct a two-class Random Forest including additional features, such as the Proximity Matrix or the Attribute Importance measure.

The *Outlier Measure* of *Random Forest* is only defined for the training data set and cannot be applied to unseen test data (5.2.6). 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. 12 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 is required.

Another useful feature which provides some insight into certain aspects of training data, is the *Attribute Importance* measureintroduced in 5.2.2. Figure 13 shows the *Attribute Importance Measures* calculated during a *Random Forest* training with 427 roller bearing normal samples.



Fig. 13 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, as can be seen in Fig, 13. The *Attribute Importance* results were used later, to retrain the *Random Forest* using only the most important attributes.

Since the built-in *Outlook Measure* is exclusively defined on the training data set, which in this semi-supervised scenario consisted of only normal condition data, a different approach had to be found for the classification problem. In this thesis, a clustering approach based on the *class* *prototype* (5.2.4) for the normal class was chosen, involving the following steps:

1. Separation of the normal feature set into a training set and a test set
2. Construction of an unsupervised *Random Forest* with the normal feature training set
3. Calculation of a normal class *prototype*
4. Calculation of the *Euclidean Distances* between the normal class *prototype* and the vectors of the normal feature training set
5. Definition of a classification threshold as the largest *Euclidean Distance* value of all the distances calculated in 4
6. Classification of test samples with a Euclidean distance to the prototype smaller than or equal to the classification threshold as normal, or as outlier otherwise.

Figure 14 shows 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 feature sets.



Fig. 14 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 normal data is enough to clearly separate normal data from outlier data in this scenario.

In a run of the method described above, the Roller Bearing normal condition feature set with 457 data vectors and 27 attributes was separated into a training set of 382 and a normal test set of 75 samples. The normal test set was then combined with 25 samples of each of the fault state feature sets, to form a complete test set comprising 150 samples.

The training set was used to construct an unsupervised *Random Forest*, including the *Variable Importance* measure and the *Proximity Matrix.* Based on the *Proximity Matrix*, the following normal class prototype was calculated:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 |

Next, the Euclidean Distances between the training vectors and the normal class prototype where calculated and the largest of these distances was used as the classification threshold. Samples from the test set with a Euclidean distance to the class prototype smaller than this threshold were classified as normal or as outlier otherwise. The confusion matrix for this classification is shown in the following table:

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***True Classes*** | | |
| ***Predicted Classes*** |  | **Outlier** | **Normal** |
| **Outlier** | 75 | 0 |
| **Normal** | 0 | 75 |

Based on the confusion matrix, the error rates were calculated as the ratios of rejected normal data (normal data classified as outliers) and the ratio of accepted outlier data (outlier data classified as normal). Both error types were 0, which proves that the simple classification method introduced here worked perfectly for the given feature sets in a semi-supervised setting.

With the *attribute importance* measures calculated after construction of the *random forest*, some of the least important attributes were removed from the data sets and several reruns of the complete classification process were conducted with the reduced data sets. The results of these reruns showed, that almost half of the attributes could be removed without a significant deterioration of the error rates

* 1. Self Organizing Maps

**Model Description**

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

The position of a neuron in the *D-dimensional* grid – or feature space – reflects the neighborhood relations in the *N-*dimensionsal data space, such that neighboring nodes in the grid are associated with neighboring prototypes in data space. With 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 in data space (van der Heijden, Duin, de Ridder, & Tax, 2004).

**Training**

Prior to the actual *SOM* training, the size and structure of the grid has to be chosen. 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 *k* denotes the number of the node and the superscript *i* specifies the iteration step in the training sequence.

1. Updating 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 how much a node is updated. The input to this weighting function is the distance of node which is updated to the winning node determined in step one. 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 defined by

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

**Classification**