Josef Suschnigg, BSc

Design and Implementation of a Predictive Maintenance System Prototype

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Dipl.-Ing. Clemens Gutschi, BSc
Institute of Engineering and Business Informatics

Dipl.-Ing. Dr. techn. Nikolaus Furian
Institute of Engineering and Business Informatics

Graz, April 2018

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date

Signature

ABSTRACT

Maintenance of production machines in manufacturing plants can be a major operative cost driver. The prediction of machine breakdowns in manufacturing environments enable the saving in costs, as well as increasing productivity by shortening reaction time for maintenance actions. This thesis describes how data can be used in an ”Industry 4.0” project and on which theoretical basis a prototype, for prediction making of machine breakdowns, can be developed for a production facility. The described time series analysis of process values and alarm messages covers feature extraction, rolling window transformation, feature selection and model building by machine learning (random forest classifier). The performance of the predictive model is depending on different parameters and is evaluated and discussed by an evaluation. The prototype, developed for Audi Hungaria Zrt., covers all necessary actions for a time series data analysis and provides results in a web interface.

KURZFASSUNG

Die Wartung von Produktionsmaschinen kann ein wesentlicher Kostentreiber in Produktionsanlagen darstellen. Die Vorhersage von Maschinenausfällen in Produktionsanlagen spart Kosten und kann die Produktivität steigern, indem die Reaktionszeit für Wartung- bzw. Reparaturarbeiten reduziert wird. Diese Arbeit beschreibt wie Daten für ein Industrie 4.0 Projekt verwendet und auf welcher theoretischen Basis eine Anwendung für die Vorhersage von zukünftige Maschinenausfälle entwickelt werden kann. Die hier behandelte Zeitreihenanalyse von Prozesswerten und Alarmmeldungen aus Produktionsmaschinen umfasst Feature Extraction, Transformation in Rolling Windows, Feature Selection und die Modellbildung durch maschinelles Lernen (Random Forest Klassifikator). Die Leistung des vorhersagenden Models, in Abhängigkeit von verschiedenen Parametern, wird anschließend in einem Evaluierungsschritt beschrieben. Der für die Motorenproduktion der Audi Hungaria Zrt. entwickelte Prototyp umfasst alle notwendigen Maßnahmen zur Datenanalyse und zeigt die Ergebnisse in einem Webinterface an.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

The engine production line of the car manufacturing facility this thesis was written for consists of many in series or parallel working autonomous machines. In Figure 3.1 a screenshot from a visualization software of an example production line with about 26 machines is shown. The unplanned breakdown of only one machine could lead to a production jam, unpredictable consequences to the work planning and result in higher costs [1]. Implementing a predictive maintenance strategy could lead to a better productivity, caused by minimized unplanned downtime and the resulting higher efficiency [2].

1.2. Goal

For this thesis a prototype, which could work in a productive environment for testing purposes, is implemented . Even so, it has a research study character and works as framework/sandbox for trying different aspects of an industrial time series analysis. It can be also used as a basis for new ideas or methods. It has a simple user interface from where a user can make predictions, on a machine subset in the production facility. Results are visualized in prediction plots and different information will be accessible from that user interface. An evaluation shows how stable the predictions are and by parameterization of different attributes of the model the influence to the performance can be figured out. In general, data analysis can be a uncertain task. From the beginning it is not clear, if the provided machine data has enough relevance to the target values and even if useful predictions can be done. The goal of this thesis is

to answer the following questions by a concrete software implementation in an automotive production environment (Audi Hungaria Zrt.):

- Is the available data suitable for predicting machine breakdowns?
- How can the implemented prototype can be improved for future work?

1.3. Thesis Structure

The written part of this thesis has a documentary character. It consists of four main sections. In the theoretical part, a general description of the whole data analysis process used for developing the prototype is covered. Also, definitions of terms frequently used in context of industrial data analysis are given. The second, practical, part shows how the prototype handles theoretical background in a concrete implementation. The outcome of this part is beside a documentation of the realization, an evaluation of the predictive model performance. These results are interpreted and possible benefits for a productive environment is derived in the third part (conclusion). The appendix provides a technical documentation, describing the prototype framework and non-obvious program parts. To collaborate with the institute the prototype source code lies on a university's GIT repository. Beside of this written document, it is the main outcome of the work.

2. Theoretical Part

The theoretical part handles the given task in general. First some maintenance types and keywords often occurring, when working with predictive maintenance, are explained. As a basis for the practical part and the implementation, the whole time series analysis process for continuous alarm messages and process values is described. At the end an approach on how to evaluate the predictive model is given.

2.1. Definitions

2.1.1. Maintenance Types

In this section an overview of general maintenance approaches [3] is given. This work handles one specific type: **PdM (Predictive Maintenance)**. Maintenance is the task to avoid or correct failures of machines. There are different maintenance types, which can be classified as illustrated in Figure 2.1:

- **Corrective** - reactively resolve a failure after it occurs
  - **Immediate** - react immediately on a failure
  - **Deferred** - react with some time delay on a failure
- **Preventive** - avoid that a failure occurs, before it occurs
  - **Predetermined** - act in predetermined schedules, i.e. do a maintenance action every 12 months
  - **On condition** - react on predefined machine condition indicators [4]
  - **Predictive** - continuously monitor the machine condition and predict breakdowns

This thesis focuses on preventive maintenance, or more specific on PdM with the following workflow: First collect all data from the data sources. Second extract the relevant data from the data pool to features by specific industry know-how or feature engineering. Then train the predictive model with machine learning algorithms. Applied to live data the model recognizes specific feature conditions and predicts the RUL (Remaining Useful Lifetime) of the machines, so that the maintenance provider can react if necessary. Repeat the process with new data after some time.

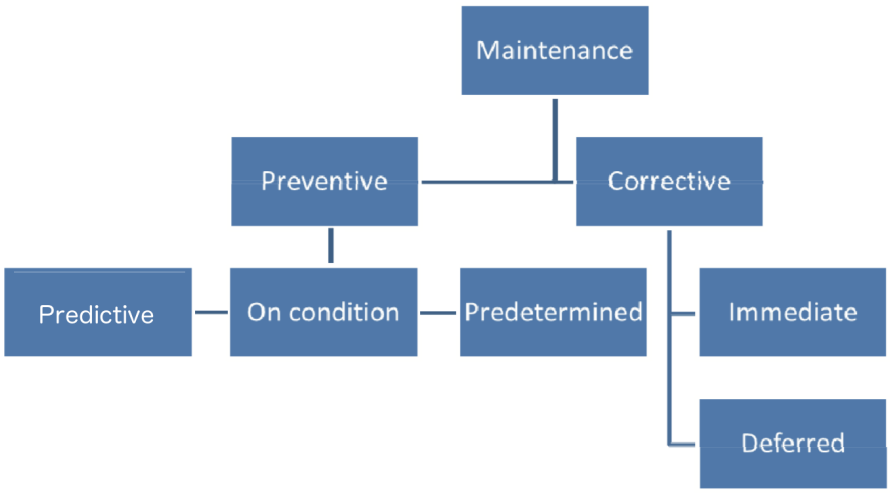

Figure 2.1.: Basic maintenance type classifications (inspired by Ojanen, 2014)

A key success factor of a predictive maintenance system is the data source quality. In subsection 2.1.4 an overview of possible data sources in manufacturing environments is described.

2.1.2. Industry 4.0

Industry 4.0 is a buzzword strongly influenced by the report of the German Industry 4.0 task force ”Forschungsunion Wirtschaft und Wissenschaft” from 2012 [5]. The document describes recommendations on how to implement the fourth industrial revolution in Germany. One difference between the first three industrial revolutions and Industry 4.0, is the attempt to define it before and not

afterwards. An essential part is the concept of the smart factory, which key concepts are shown in Figure 2.2. The description, from the report [5], about the intelligent maintenance topic in smart factories, freely translated (from German), is:

> *The maintenance management will be self organized. Unplanned events (machine failure, quality fluctuations or changes of product specifications) will be automatically identified. Condition and wear of materials will be continuously monitored and predicted. Through this adaptation of the whole production process, unplanned machine downtime will be avoided.*

By this definition, the work of this thesis can be part of an ”Industry 4.0” smart factory. In the report the importance of data management and the technology trend ”Big Data” is mentioned. Certainly many data is needed for a predictive maintenance project, but what is ”Big Data”?

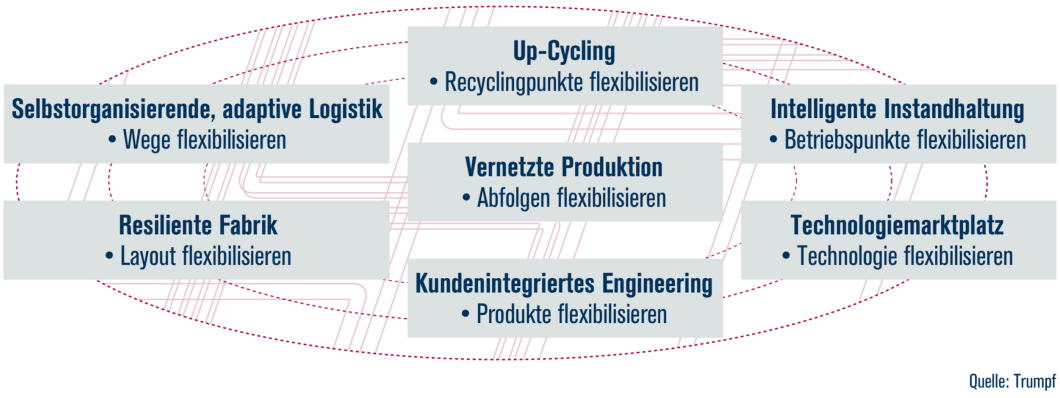

Figure 2.2.: Main topics of the Industry 4.0 Smart Factory

2.1.3. Big Data

During research on this thesis the keyword ”Big Data” occurred many times in context of PdM. According to the META Group (now Garnter) [6] there are three criteria, also called the ”3V”s, to fulfill the attribute ”Big Data”:

- **Volume**: The volume of datasets is too high to be processed by common computer systems.
- **Velocity**: The data is generated in real time.
- **Variety**: The data is unstructured and complex.

The detailed description and definition of the data used in this project follows at section 3.2. From there it is clearly recognizable that neither the **volume** nor the **variety** criteria are satisfied. The data is not well-structured, but in general, every data record can be assigned to a specific node (machine) and a concrete time-stamp. Also, the volume is not that big, that some special software for clustering or distributed computing is required. Anyway the **velocity** is high enough, since the data is generated continuously. So at least data used in this work, fulfills one criterion of the ”3V”s.

2.1.4. Computer Integrated Manufacturing

CIM is a term firstly appeared in the early 1970s and describes how software and information systems can support manufacturing. Since then 37 different and similar models of CIM appeared [7]. In Scheer’s Y-Model [8] the connection between technical (computer aided CA*), production planning and controlling information systems are shown. For the prototype, that means, every information system used to support in a manufacturing process can be a possible data source for a predictive maintenance system (in theory). All data sources used for the prototype of this thesis are part of the model shown and are highlighted (green) in Figure 2.3:

- **ODC, Operational Data Collection (de: BDE)**: Log of machine state changes
- **Maintenance (de: Instandhaltung)**: Data collection of maintenance actions, which have been taken
- **Work Management (de: Arbeitsplanung)**: Working schedule of the manufacturing plant

- **CAQ, Computer Aided Quality (de: Qualitätssicherung)**: Continuous machine sensor data

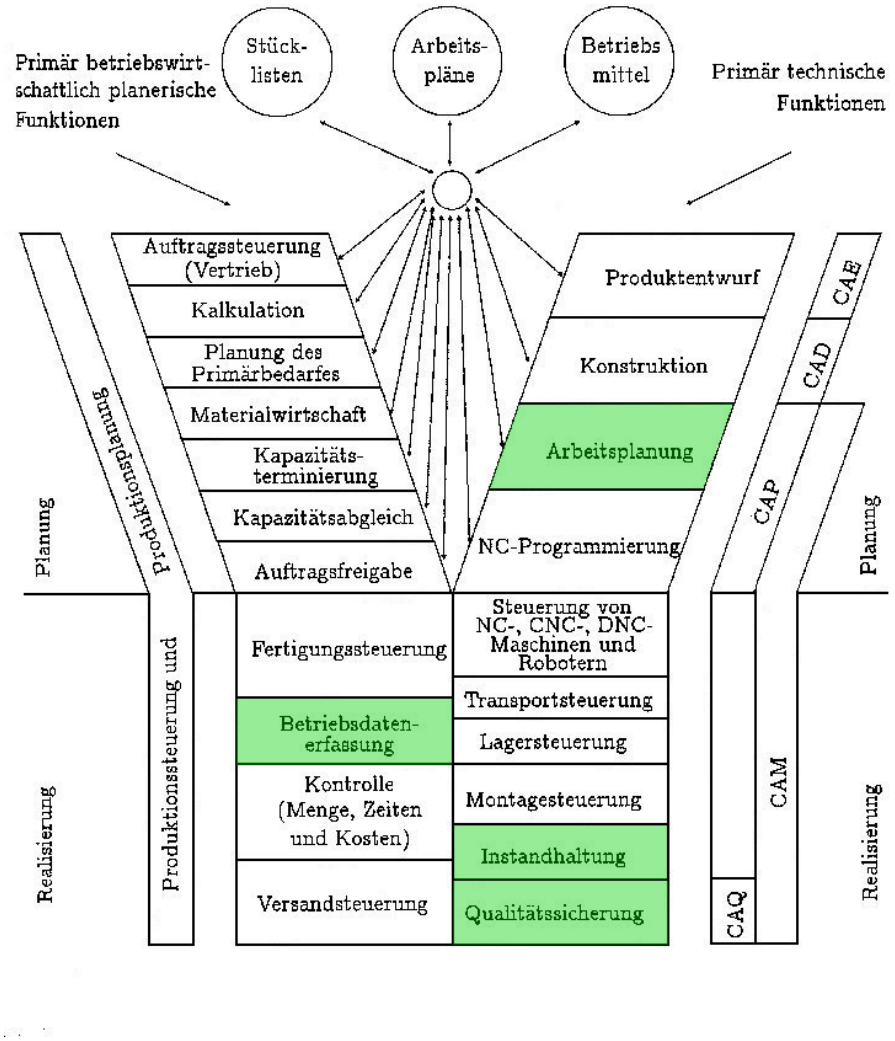

Figure 2.3.: Computer Integrated Manufacturing by Scheer (de, 1987)

Different data sources, database tables and entity models are shown and explained more detailed in section 3.2.

2.2. Data Analysis

In Figure 2.4 the whole data analysis process for this problem is illustrated. First the data processing step extracts features from a data collection or a data

stream to observation and rolling window features. After that, a feature subset of the best extracted features is chosen. Then an appropriate predictive machine learning model is trained and evaluated. If the model performance is not sufficient, the feature selection and modeling step can be refined in an evaluation step. The analysis results in a model with suitable performance and can be used in a productive environment.

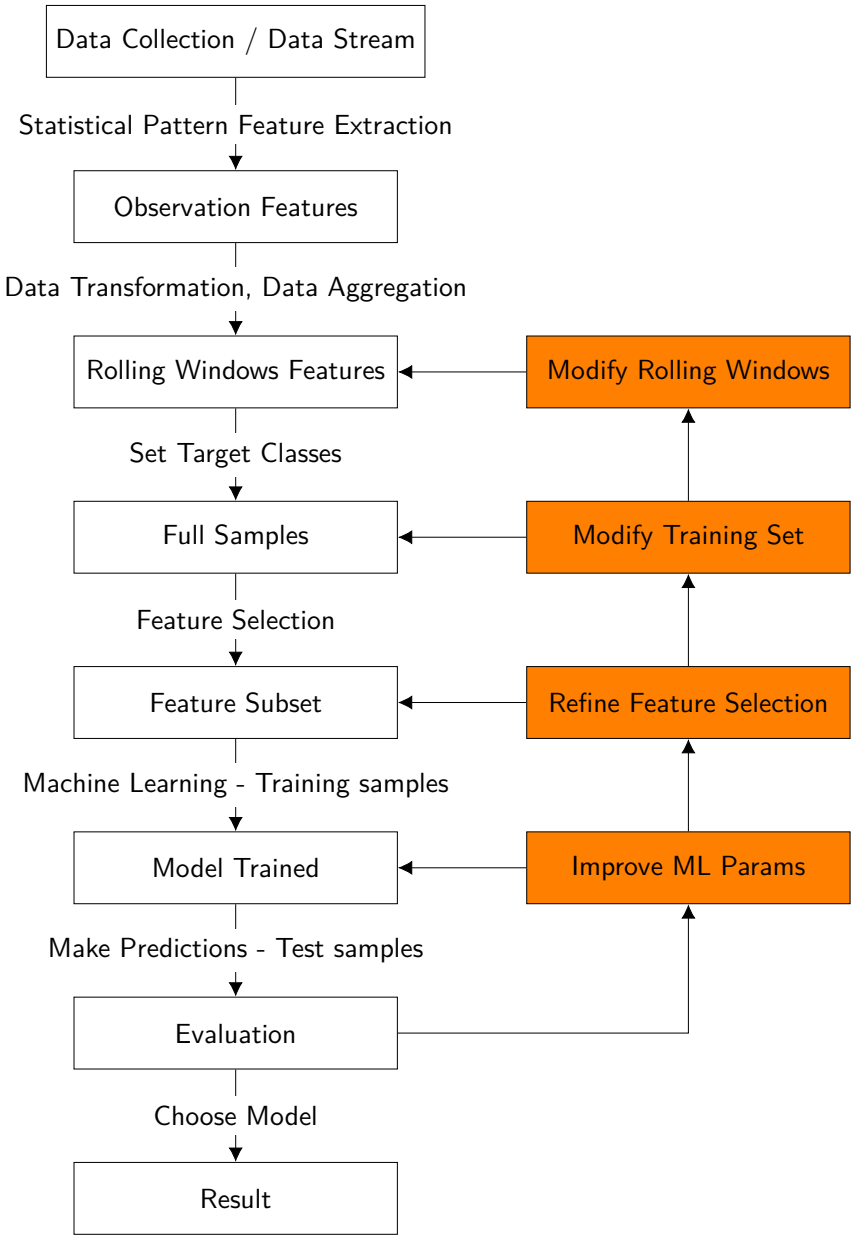

Figure 2.4.: Data analysis scheme for this work

2.3. Data Processing

The goal of data processing is to transform data into a format supervised learning algorithms can handle, like defined in Equation 2.1.

$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(n)} \\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(n)} \\ \vdots & \vdots & \dots & \vdots \\ x_m^{(1)} & x_m^{(2)} & \dots & x_m^{(n)} \end{bmatrix}, y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \tag{2.1}$$

where m is the number of samples and n the count of features. $y \in C$, whereas C is the set of c target classes:

$$C = \{c_1, c_2, \dots, c_c\} \tag{2.2}$$

For example the j^{th} input vector is defined as $\{x_j, y_j\}$:

$$\{x_j, y_j\} = \{x_j^{(1)}, x_j^{(2)}, \dots, x_j^{(n)}, y_j\} \tag{2.3}$$

The performance of trained models depends on the quality of input data X and the target data y. Instead of using classes C, also continuous values for regression can be used.
In the next sections training data is extracted to features by statistical patterns and rolling time windows.

Observation Feature Extraction

To split all available data of a time window t, into processable parts, observations are introduced. An observation O collects related data (i.e. data of one specific machine) of a predefined time window and transforms it by feature extraction into features. The length of an observation time window depends on the goal to achieve and the nature of the task domain itself. The time window size should not be too small, so that any useful information will lose significance, but also not too big, so that the observable features become fuzzy.

By statistical feature extraction, related data signals occurring in observations are transformed to features [9]. In this thesis one of the following statistical measures, for a series of signals $s_1, s_2, ..., s_n$ are used to extract features of an observation data channel:

Arithmetic mean The mean, in an observation time window, is calculated by:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} s_i \tag{2.4}$$

Count occurrences Number of occurrences of a signal:

$$c = n \tag{2.5}$$

Value sum Sum of signal values:

$$s = \sum_{i=1}^{n} s_i \tag{2.6}$$

Minimum signal value The lowest numerical value of a signal:

$$min = min(s_1, s_2, s_3, ..., s_n) \tag{2.7}$$

Maximum signal value The highest numerical value of a signal:

$$max = max(s_1, s_2, s_3, ..., s_n) \tag{2.8}$$

Difference count The count of difference between two ascending successive signal values (i.e. used for counting the number of produced items in a time frame, if the signal is a counter).

$$d = \sum_{i=2}^{n} s_i - s_{i-1} \tag{2.9}$$

The result of observation feature extraction is a series of observations $O_1, O_2, ..., O_o$. An observation O, collects data of a predefined time frame and has n statistical features:

$$O = feature_1, feature_2, ..., feature_n \tag{2.10}$$

The explained method is illustrated in Figure 2.5.

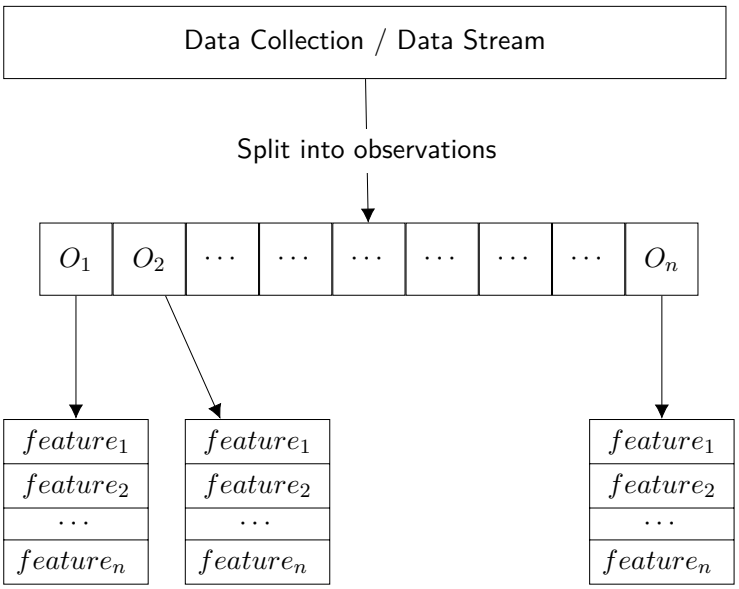

Figure 2.5.: Extract observation features from a data collection or data stream

2.4. Rolling windows

After splitting the related data channels into observations, rolling windows are created. An observation contains information of a relatively small time frame. To create meaningful input samples, information of previous episodes are necessary. So complete sample will consist of $w_0, w_1, ..., w_W$ not overlapping rolling windows, whereas one window aggregates features of all observations included, as illustrated in Figure 2.6. The aggregation calculates either the mean or max value, of observations, like explained in section 2.3.

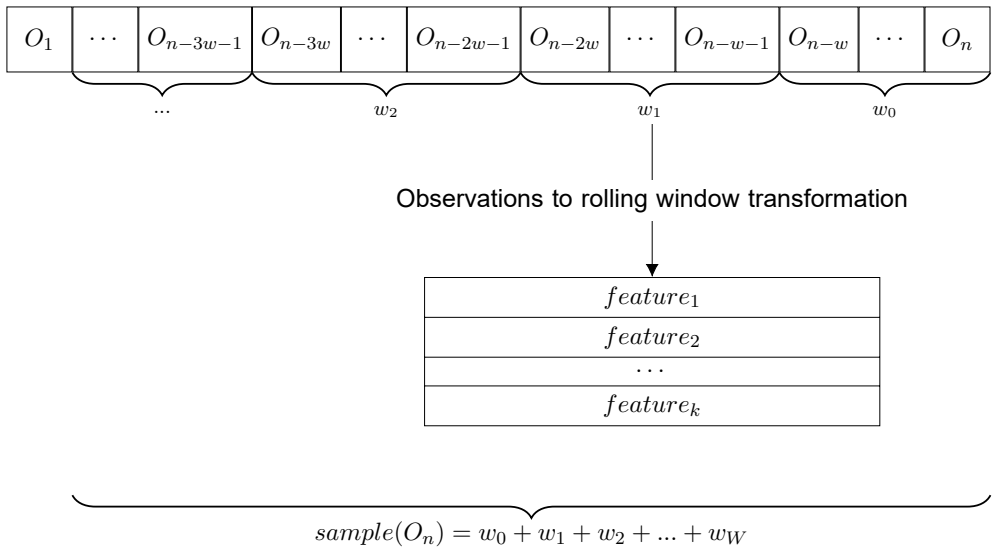

Figure 2.6.: Create sample from observations and rolling windows for O_n

2.5. Setting target values

For predictive maintenance, target values of a sample may contain the RUL in hours, which in general describes the duration till the next incident or machine failure occurs. Because regression models may be too specific to create meaningful predictions, it is converted into a classification problem. According to Equation 2.2 classes are needed to be defined. Each class describes a RUL specified time frame. For example, the RUL of 14 days can be split into seven classes, where each class covers a time frame of two days, as shown in Table 2.1.

Table 2.1.: RUL to class conversion example

RUL class	**RUL time frame**
0	not defined / fallback
1	0 - 24 hours
2	24 - 48 hours
3	48 - 84 hours
4	84 - 132 hours
5	132 - 216 hours
6	216 - 300 hours
7	> 300 hours

2.6. Full samples

After feature extraction and target value setting, samples for each observation window are complete and can be put into input vectors as shown in Equation 2.1. The result of the process to this point are m samples (depends on the size of the time window the data is extracted from), where to each sample a target class c is assigned. Each sample is a rolling window feature vector and has the form of:

$$x = \begin{bmatrix} w0_1 & \cdots & w0_k & w1_1 & \cdots & w1_k & \cdots & wW_1 \cdots & wW_k \end{bmatrix} \tag{2.11}$$

The data collection has been prepared for the next steps and is ready for the machine learning algorithm to train a predictive model.

2.7. Feature selection

Not all extracted features are appropriate or necessary for building predictive models. Choosing a subset of features can have multiple advantages. For example, by removing redundant features (i.e. by reason of no variance), the time needed to compute models and to predict can be improved. Also, for a better accuracy, feature selection may be needed to reduce noise.
There are several methods and algorithms for this purpose. In this thesis, features are selected by calculating the Gini index, described in subsection 2.8.2. The information gain of features depends on the target value each sample. In this predictive maintenance task the strength of single features, strongly differs by varying the incident target values.
For example a feature could be high on information for one incident, but completely irrelevant for another one. A general approach, of simply using every incident may not work, because of incomplete maintenance/incident documentation. Also, most of the observation may not show a significant change before each documented machine failure, because failures are not properly classified. Overall feature selection, is done by choosing different incidents, when setting

target values as described in section 2.5 and the entropy calculated by the Gini index.

2.8. Machine Learning Algorithm: Random Forest

To understand the random forest algorithm, classification and regression trees (CART) [10] and the Gini index needs to be explained first.

2.8.1. Classification and Regression Trees - CART

CARTs are decision trees to model classification or regression prediction problems. Every tree node models a feature of an input vector and leafs represent target values. In this thesis an input vector uses classified RUL as target values and models a classification problem, as described in section 2.5.

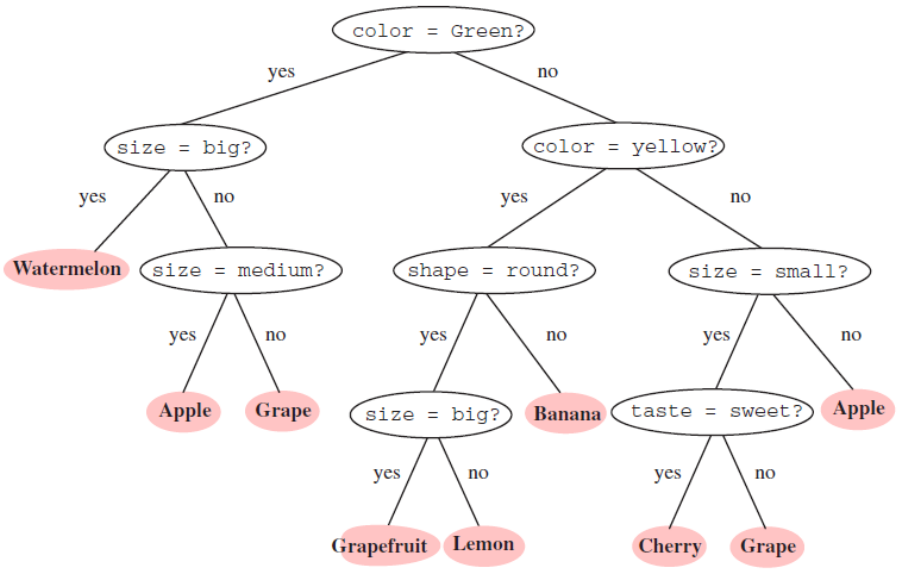

Figure 2.7.: Example fruit classification tree model

Source: http://www.narendranaidu.com/2014/02/ruminating-on-decision-trees.html, 3/10/2017

In Figure 2.7 an example for a fruit classifier tree is illustrated. It has four features (color, size, shape and taste), seven classes (watermelon, apple, grape, grapefruit, lemon, banana and cherry) and eight decision points, also called splits or nodes. When feeding the model with an input vector, containing four

feature values, the prediction tree outputs one specific class. In contrast to many other machine learning algorithms, CART models are understandable and one can easily trace back the decision-making and identify important or strong features. Also, the computation time is limited to the number of features and the resulting tree height, which means even with lots of features the algorithm should runs fast.

> For completeness: Using CART for a regression, is almost identical to the discrete value classifier. The difference is, that when training the model, multiple target values can be part of one specific permutation of decisions in the tree. To get the real (continuous) number of such a ”class” the mean squared error of the corresponding target values is calculated.

2.8.2. Feature Split - The Gini index

Class	No	No	No	Yes	Yes	Yes	No	No	No	No
	Annual Income									
Sorted Values →	60	70	75	85	90	95	100	120	125	220
Split Positions →	55	65	72	80	87	92	97	110	122	172	230
	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >
Yes	0 3	0 3	0 3	0 3	1 2	2 1	3 0	3 0	3 0	3 0	3 0
No	0 7	1 6	2 5	3 4	3 4	3 4	3 4	4 3	5 2	6 1	7 0
Gini	0.420	0.400	0.375	0.343	0.417	0.400	*0.300*	0.343	0.375	0.400	0.420

Figure 2.8.: Example Gini index calculation, from [11] p. 162

Every node in a CART tree models a feature which can be either a class or a continuous number. Somehow a value needs to be found which splits all different values for a feature of a training set into two groups (binary tree). The measure to find the best split are often based on the degree of impurity, which is an indicator for unequal distribution. For example a node with class distribution (0,1) has maximum impurity, whereas a node with uniform class distribution (0.5, 0.5) has the zero impurity [11]. The measure of impurity used

in this work is the Gini index:

$$Gini(t) = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2 \tag{2.12}$$

, where $p(i|t)$ is the fraction of records belonging to feature i and class t , and c is the number of classes. In Figure 2.8 an example is given on how to calculate the Gini-index on continuous data and two classes. The best split position is the value with the lowest Gini coefficient. In the example, a hypothetical feature node which models annual income, will be split into ”<=97” and ”>97”, caused by the lowest Gini index ”0.300” (underlined).

2.8.3. Random Forest

Random forest is a supervised machine learning algorithm. A forest consists of many CART trees and according to Breiman [12] the algorithm processes the following steps [13]:

1. Choose n, the amount of trees in the forest, and $m < M$ features every tree should have, where M is the number of features in the input vector.
2. For every tree choose m features randomly.
3. Calculate the best split positions for every node (Gini coefficient).
4. Fully grow the tree and continue at 2 till the forest has n trees.
5. To classify an input vector, every tree votes for a class. Output of the forest is the class with the most votes.
6. Random forest can also return the probability of a class, by diving the number of trees, voted for a class with the amount of trees in the forest. This can be useful, when a class probability exceeds a significant threshold, but usually would not be the resulting class of the random forest model.

2.9. Training Strategy

To measure the performance of samples, a strategy is needed. The outcome of experiments, during this work, have shown that simply splitting the whole sample set, for a standard validation approach (like k-fold [14]), is not effective for the targeted predictive maintenance task. Not all the extracted features occur before an machine failure. Also, due incomplete maintenance recordings, there is a high chance, that the RUL classes, of the samples are wrong. To ensure that at least the targets of a training set are correct, a small subset of samples needs to be chosen. The following incident based training set strategy has been implemented for the prototype:

1. Define a time frame, previous of an incident to pick samples from. It is necessary, that all target classes are at least in on sample included. Otherwise, the resulting model can't predict the missing classes.
2. Choose documented incidents, where the engineered features show a significant change.
3. Add samples located in the specified time window of all chosen incidents to the training set. To ensure uniform a priori probability of the target classes, it is needed that they are equally distributed in the training set.

When the training set is complete, the chosen machine learning algorithm can train a predictive model.

2.10. Ensemble Prediction

After modeling the classifier, it can predict a target class for any sample, having the same dimensions as the training samples. During work on this thesis different approaches for predicting have been tried:

- **Simple prediction:** For every test sample in a time series, return a target class. The RUL for a specific moment can be mapped from the output class and the RUL definitions (Table 2.1).

- **Probability prediction:** The simple approach appeared to be too general and not meaningful for productive usage. Classifiers, used in this work, are able to return the probability of every target class. To utilize this feature, an approach has been tried to visualize the probabilities of predicting a specific class in a time series, by plotting the predictions to a resulting prediction curve. The left plot in Figure 2.9 illustrates this approach. It can be seen that the curve is not steady and is likely to jump between neighbored observations.

- **Ensemble prediction:** This approach is illustrated on the right plot of Figure 2.9. It can be seen as an improvement of probability prediction. The RUL curve is more stable and also multiple previous predictions are included in the probability calculation. It is easier to interpret and will be used in the practical part. In the following section an algorithm for this approach is explained.

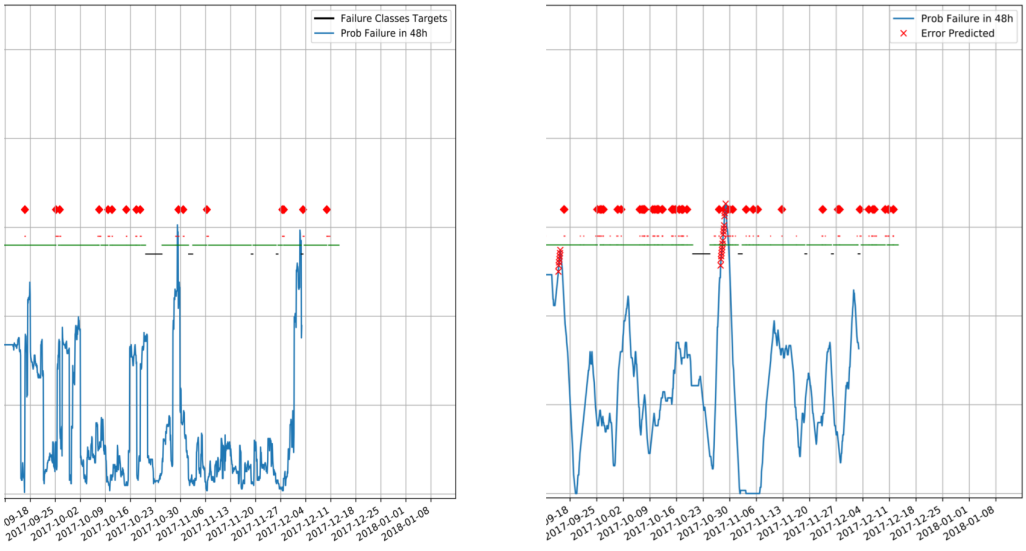

Figure 2.9.: Comparison between probability prediction (left) and ensemble prediction (right) for a machine failure in 48 hours.

2.10.1. Ensemble prediction algorithm

Inspired by weather ensemble forecasting techniques [15] and moving average [16], prediction in this work are calculated by an ensemble of predictions. When making predictions for the probability of machine failures, it can be useful that the resulting curve is steady. Primary to perceive a trend and derive maintenance actions from it. The ensemble prediction approach predicts the probability of a specific RUL class on a single observation. To make the prediction more stable also the predictions of previous observations are included. The algorithms for ensemble prediction covers the following steps:

1. Define a number e of observations, which should be included in an ensemble.
2. Choose a class c, the probability should be predicted for.
3. Every sample in the testing window makes a prediction and stores the resulting class in a data structure.
4. Starting with the e^{th} sample, make an ensemble prediction for every observation :
    a) Get previous e samples for the current observation and add them to an ensemble.
    b) For the first sample, set class c as the target class and the according RUL in hours.
    c) For every other sample, set RUL hours based on the RUL of the first sample and the time difference between the actual and first sample.
    d) Set the target class for every sample based on the calculated RUL.
    e) For each sample in the ensemble, compare the target class and the predicted class and if they are equal increment a counter.
    f) Divide the count by e to get the probability for a target class of a single observation.

In Table 2.2 an exemplary execution of the algorithm is shown. It predicts the probability of RUL class 2 for the 55th sample with 65.00%. When predicting the probability for class 2 of the 56th sample, the algorithm would replace the 16th sample with the 57th in the ensemble. The similarity of neighbored observations will produce similar probabilities, and a steady RUL curve.

2.11. Evaluation

To measure the performance of a model evaluation is needed. Typically, this is done on a test sample set, which covers unseen data. Meaning, no sample used for training is used for testing. A common method for evaluation are confusion matrices. For every sample in the test set a prediction is made. Depending on the target and predicted class, either one of the following evaluation classes is chosen:

- **True Positive:** Both the prediction and real class reveal an error.
- **False Positive:** The prediction indicated an error, but the real class is not.
- **True Negative:** Either the prediction and real class, reveal no error.
- **False Negative:** The predicted class indicates no error, but the real class does.

Prediction \ Real class	c	c^c
c	true positive (tp)	false positive (fp)
c^c	false negative (fn)	true negative (tn)

Figure 2.10.: Confusion Matrix.

Source: Prof. Denis Helic “Knowledge Discovery and Data Mining“ lecture slides, Winter term 2017/18

The resulting evaluation classes of the test set is summarized in the confusion matrix, shown in Figure 2.10. For the specific predictive maintenance task of

Table 2.2.: Exemplary ensemble prediction for sample 55: $e = 40$, $c = 2$

Sample	RUL	Target	Prediction	Count	Probability c
55	48	2	2		
54	50	2	3		
53	52	2	1		
52	54	2	4		
51	56	2	4		
50	58	2	4		
49	60	2	4		
48	62	2	2		
47	64	2	2		
46	66	2	2		
45	68	2	2		
44	70	2	2		
43	72	2	2		
42	74	2	2		
41	76	2	1		
40	78	2	2		
39	80	2	3		
38	82	2	2		
37	84	2	2		
36	86	2	2		
35	88	3	2	Equal Classes = 26	
34	90	3	2		26 / 40 = 0.65
33	92	3	2		
32	94	3	2		
31	96	3	3		
30	98	3	3		
29	100	3	3		
28	102	3	3		
27	104	3	3		
26	106	3	3		
25	108	3	4		
24	110	3	3		
23	112	3	6		
22	114	3	3		
21	116	3	5		
20	118	3	3		
19	120	3	3		
18	122	3	2		
17	124	3	2		
16	126	3	1		

this thesis, only one target class is evaluated and a partial confusion matrix is build. The mapping of ensemble prediction class probabilities to evaluation classes is done as:

- Define a probability threshold t first. For example $t = 0.50$ can be appropriate.
- If the predicted probability exceeds t, is higher than the probability of the previous sample and the curve rises for a specific amount of ticks, indicate the prediction as positive.
- All other predictions are negatives.

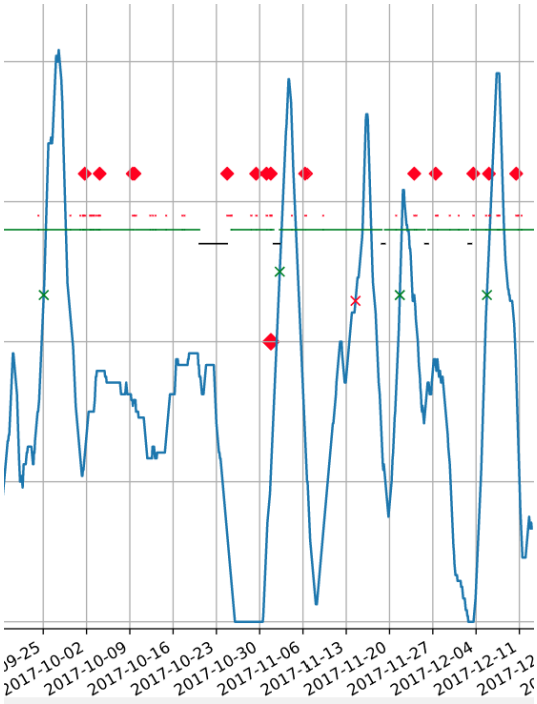

Figure 2.11.: Example positives. Red crosses on the blue prediction curve indicate false positives, green ones true positives.

The following algorithm checks if a sample positive is true or false:

1. Find all errors in a time frame beginning at the sample time plus two times hours of the RUL class. From now this time frame is called tolerance band.

2. If no error is included in the time frame, the positive is false, true otherwise.

In Figure 2.11 the assignment of negatives and positives is illustrated. The blue line shows the ensemble predictions of a failure class over a time series. If the curve rises and exceeds the threshold of 50 % a red cross indicates a predicted positive. During this work it turned out, a complete confusion matrix is not the best way to evaluate the model's performance. It is possible but not the most expedient solution. Visualizing the prediction curve as shown in Figure 2.11 can be more meaningful than a simple numerical evaluation. From experience during development, a rising prediction curve indicates a dropping machine health, in many cases leading to a machine error. Due incomplete or missing failure documentation the uncertainty of the meaning of the data, can lead to incorrect true and false negatives. To avoid this limitation on certainty in data, for evaluation only true and false positives will be included.

2.11.1. Evaluation Indicators

Additionally to the partial confusion matrix (true and false positives) the following indicators for evaluation are introduced.

- **True positives (tp):** Number of correct hits
- **False positives (fp):** Number of wrong hits
- **Precision:** $\frac{tp}{tp+fp}$
- **Mean Positive Accuracy:** value between 0 and 1, average positive accuracy of all errors (see next section)
- **Mean Negative Accuracy:** value between -1 and 0, average negative accuracy of all errors (see next section)
- **Max Positive Accuracy:** value between 0 and 1, maximum positive accuracy value of all found errors (see next section)

- **Max Negative Accuracy:** value between -1 and 0, maximum negative accuracy value of all found errors (see next section)

Prediction accuracy

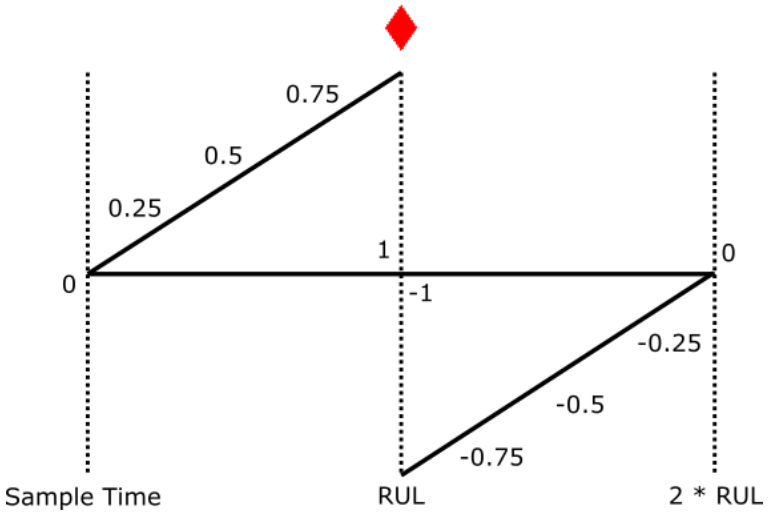

Figure 2.12.: Prediction accuracy tolerance band. The left zero is at the sample time position, and the right zero at sample time plus two RUL. Best accuracy is given in the center where the red diamond indicates an error.

As explained before the tolerance band catches every error occurring in an double RUL time frame. To give an indicator on how well the errors are found an average of the position of the errors in the tolerance band is calculated. The positive accuracy value indicated the position of errors, which have been found too early and the negative accuracy indicates errors found later than the actual predictions RUL. When many failures occur in the tolerance band, these values can't become the maximum of 1 (100%). So additionally either the maximum values for the negative and positive area is added as an evaluation indicator. This metrics should give the information if threshold t needs to be changed, whereas a low positive accuracy indicates t being too small and a high negative accuracy indicated t being to high. The best case would be a single error lying in the middle tolerance band, indicating an positive accuracy of 1 and a negative accuracy of -1. Figure 2.12 illustrates the tolerance band

and the according accuracy values (from -1 to 1) for every error included.
The listed indicators are used to measure the performance of models, and to make them comparable. It should not be seen as an absolute value, but sufficient for comparing different models. In the next sections parameters are described on how to change models.

2.12. Model Improvement

Like illustrated in Figure 2.4, after the evaluation step there are four ways outlined (orange boxes) to improve the model with further iterations. The following are used for this work:

2.12.1. Improve machine learning algorithms parameters

Every machine learning algorithm has parameters to influence the performance of resulting model. They are also suitable to prevent over- and under-fitting. The algorithm used in this thesis is random forest and several parameters are available to control model training.

2.12.2. Select different features

The feature selection is done by calculating the Gini index and choosing the best ranked ones. Setting different target values for samples (i.e. other incident classes) results in diverse rankings for feature selection. So for evaluation purposes, improvement can be achieved by varying target values and the training set.

2.12.3. Evaluation

The main parameter in the evaluation method is the threshold t. It influences, when the prediction curve evaluates a positive.

2.12.4. Choose different ensemble window predictors

Varying size of the ensemble prediction windows, may influence the resulting prediction curve. Also modifying the design (i. e. choose every x^{th} predictor) is evaluated.

All parameters used in the practical part for evaluation are listed in Table 2.3:

Table 2.3.: Evaluation parameters

Step	**Parameter**	**Description**
Random Forest	num_estimators min_samples_leaf	number of trees in a forest min samples for a leaf
Evaluation	threshold t	modify threshold percentage
Ensemble Window	vary predictors	modify size and distribution of predictors

3. Practical Part

The practical part shows an implementation of the concepts and methods described in the theoretical part. Technical details of the implementation (libraries, databases, frameworks, source code) are specified in greater detail in the technical documentation. Even tough explanations to the configuration file of the software, are given in this chapter and the relevant sections. They appear in the form:

```
In the ''smada'' directory of the Django framework a configuration
    file config.py is located. Variables and their meanings are
    explained in the relevant sections.
```

3.1. Audi Hungaria Zrt.

Audi Hungaria Zrt. is a car engine manufacturing plant in Györ, Hungary, about 100 km to the east from Vienna. With about two million produced engines in 2016 the location is one of the biggest engine manufacturer worldwide. There are about 2500 machines in operation, producing up to 8,800 engines per day, and every of them need maintenance actions [17]. Audi is already collecting data by their PLC—based ODC solutions and also by their ERP system. Using this data sources, this work implements a prototype predictive maintenance system for a production line with about 26 machines.

The data analysis process planned at an early phase of this work, illustrated in Figure 3.2, covers all single steps needed and will be described in this chapter. Data analysis is an uncertain task, what makes exact planning at the beginning

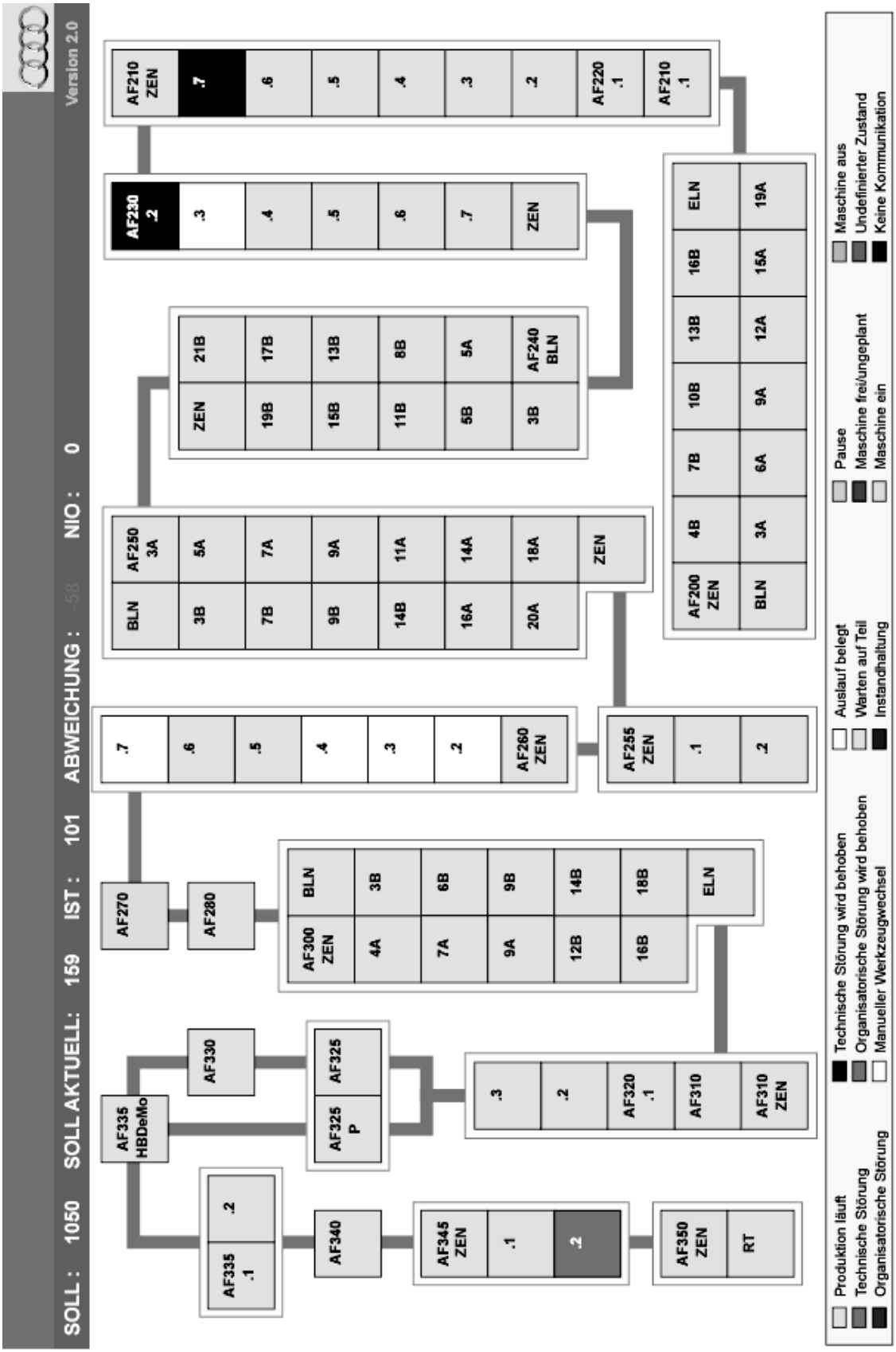

Figure 3.1.: Example Production Line (Screenshot of Visualization Software)

difficult. So, the process slightly differs from the final implementation.

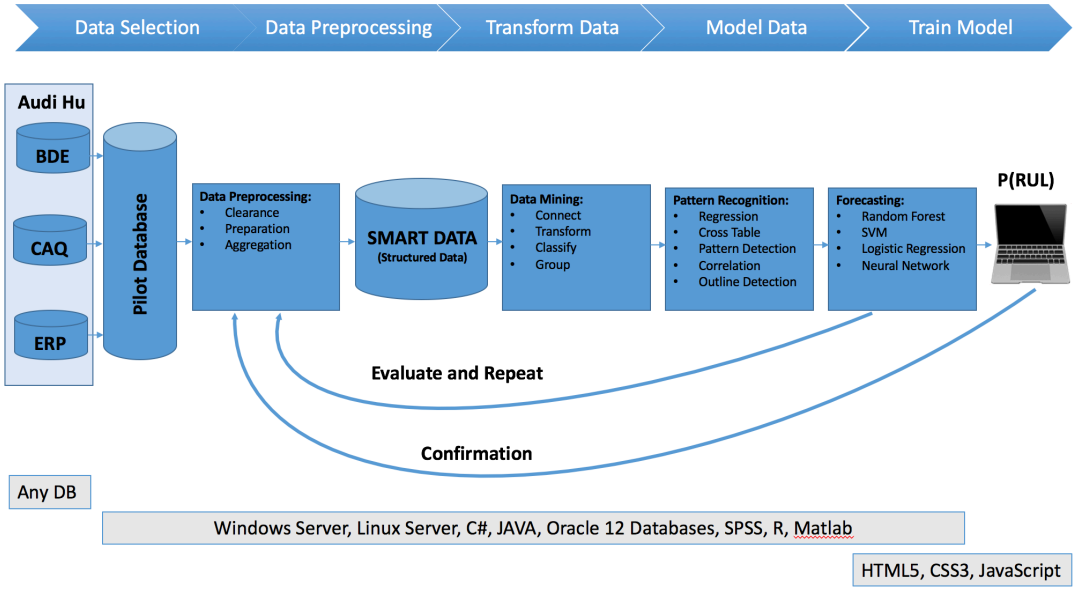

Figure 3.2.: Predictive maintenance scheme created for Audi Hungaria Zrt.

3.2. Data Collection

As mentioned subsection 2.1.4 there are many possible data sources in a CIM environment. But not all of them are suitable or accessible for a PdM task. The prototype will use the following information systems:

ODC Machines are able to log two different types of operation data information. First there are about 30 machine states (see Figure 3.4) and secondly almost about 1 000 000 different alarm messages. Both are predefined by the machine manufacturer and find its way to the database through the PLC system.

CAQ This information system stores continuous data received by sensors, and counters. For example the total quantity of produced items, the total quantity by item type, the active energy and effective power usage of the PLC will be tracked by CAQ.

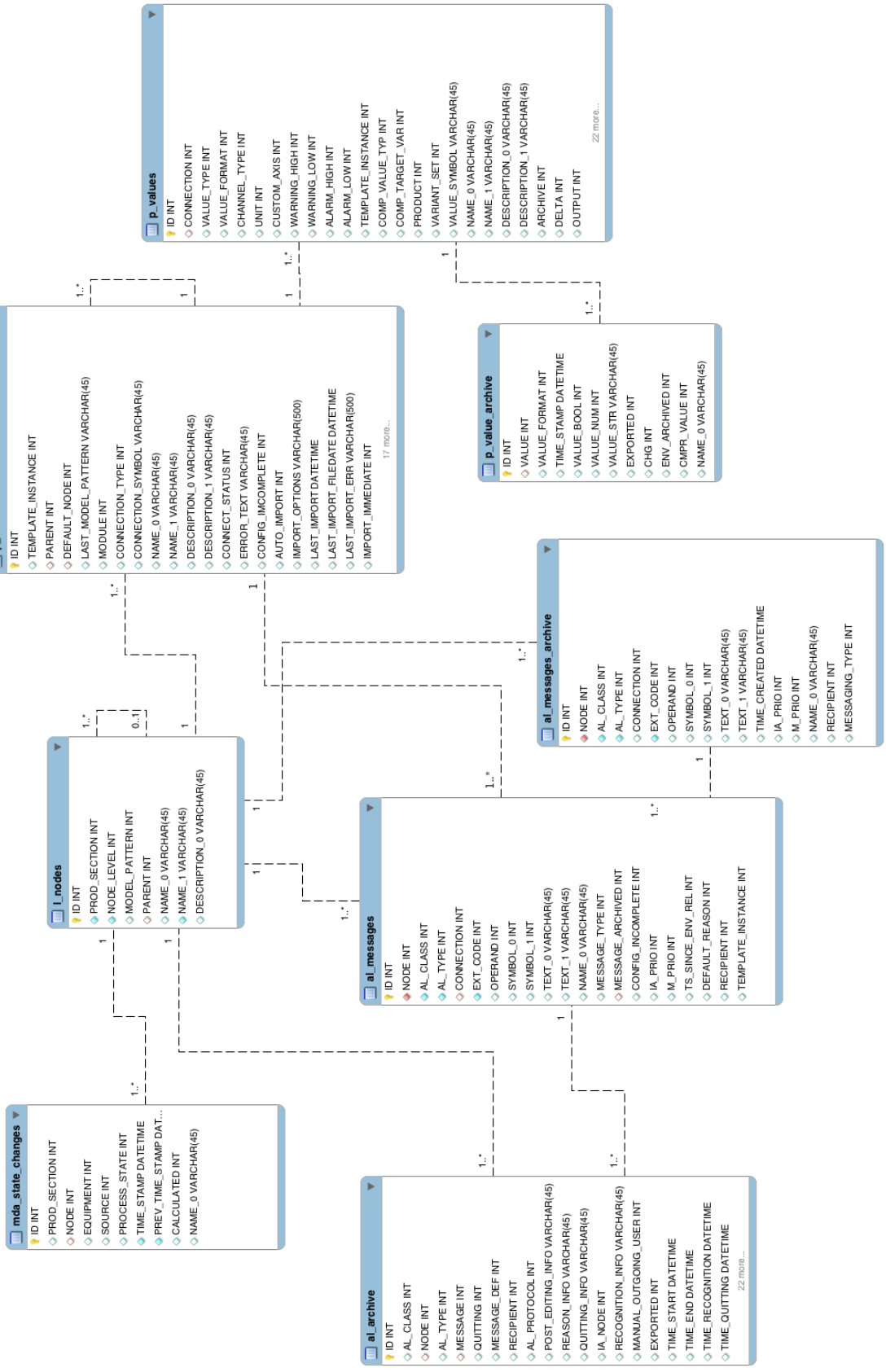

Figure 3.3.: Raw Data Entity Relationship Diagram

ERP In theory, there are at least two types of information, which can be extracted from the ERP system. First, the shift calendar, which represents the working schedule of each machine to identify the machines planned uptime, can be identified. And second, maintenance actions, which happened, are recorded here. These are the target values our prototype will be trained to. As explained below, in practice this data is generated by a workaround (shift calendar).

All the data described above is part of the pilot database and will be directly offered by Audi. Since the project this thesis was written for is called ”SMADA - Smart Data Analyser”, this data collection used for developing the prototype is named ”Raw Data”.
In this section a detailed description of the raw data database is given. Figure 3.3 shows the related ER-model. Some tables are too extensive, so a few field definitions are collapsed in the figure. All database tables, but only its important **fields** are explained here:

ODC Database Tables

Machines: l_nodes The nodes database table models production machines and its relations to each other. Nodes have a three structure with eight levels. **End nodes (machines)** which have the same parent are part of the same working procedure, production line or are somehow related.

Table 3.1.: l_nodes relevant fields

field name	description
id	unique identifier of the node
node_level	depth level of the node in the node tree
parent	parent node one level upwards
name_0	name of the unit
description_0	optional description of the machine
sort_id	order of nodes which have the same parent

Machine state changes: mda_state_changes Every end node has a specific state at any time. All state changes, occurring over time, are recorded in

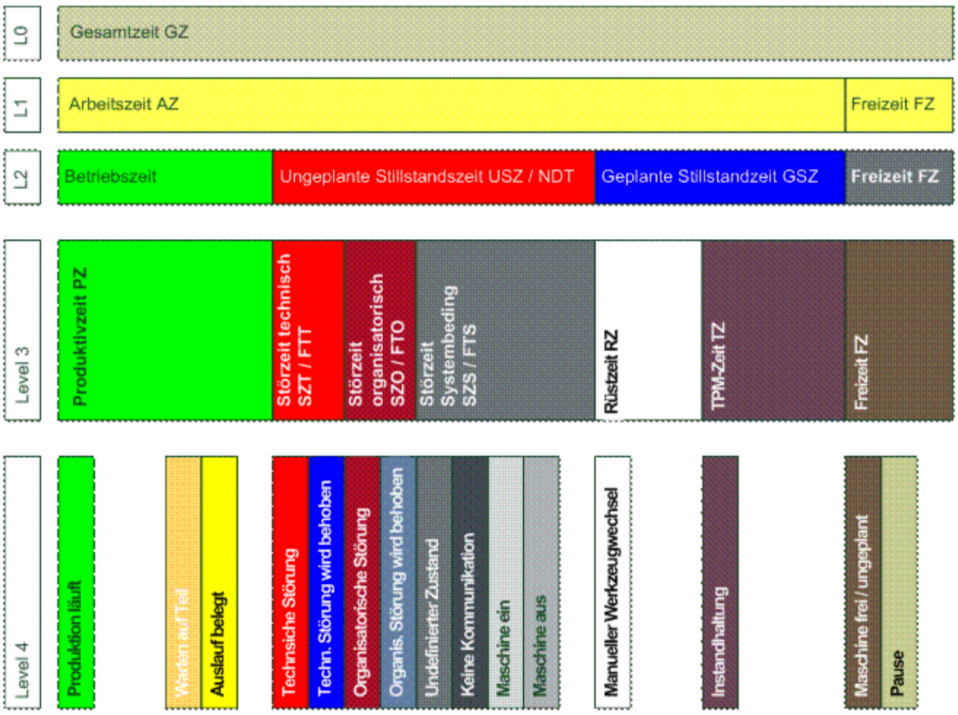

Figure 3.4.: Tree structure of machine states

Table 3.2.

Table 3.2.: mda_state_changes relevant fields

field name	description
id	unique identifier of the entry
node	id of the associated end node (l_nodes)
process_state	id of the associated state (mda_state)
time_stamp	time when the state ends
prev_time_stamp	time when the state begins

Table 3.3.: mda_states relevant fields

field name	description
id	unique identifier of the state
lvl	level in the state tree
parent	superior state
symbol	shortcut of the state
name_0	German state name
color	color hex code of the state
sort_id	order of states with the same parent

Machine State Definitions: mda_states This database table is static and contains all possible states a machine can have. They imply a tree structure which means, that a state can have one or more higher leveled states. The structure is illustrated in Figure 3.4.

CAQ Database Tables

400.000 - 499.999	Allgemeine PLC-Alarme	
500.000 - 599.999	Kanal-Alarme	Die PLC-Alarme im Bereich 500.000 - 899.999 werden vom Maschinenhersteller projektiert und beschrieben
600.000 - 699.999	Achs-/Spindel-Alarme	
700.000 - 799.999	Anwenderbereich	
800.000 - 899.999	Ablaufketten/Graphen	
	(810.001 - 810.009 Systemfehlermeldungen in der PLC [1)])	
900.000 - 965.999	HMI PRO sl Runtime	
966.000 - 999.999	Reserviert	

Figure 3.5.: Example mapping of alarm messages from the PLC manual

Source: Siemens

Message Definitions: al_messages The control software of production machines can return about 1 000 000 different exit codes. In Figure 3.5 the definition of some exit alarms classes and their identifiers is shown. In the al_messages database table alarms and exit codes are stored.

Table 3.4.: al_messages relevant fields

field name	description
id	unique identifier of the entry
node	id of the associated end node (l_nodes)
connection	id of the associated connection (p_connections)
message_archived	id of the associated archived message (al_archive)
ext_code	PLCs exit code

Message Archive: al_archive This database table is used to assign an alarm message to a specific start and end time.

Process Connections: p_connections The CAQ system does not use the l_nodes table to identify machines. Instead, it uses the process connections

Table 3.5.: al_archive relevant fields

field name	description
id	unique identifier of the entry
node	id of the associated end node (l_nodes)
message_def	id of the associated state (al_messages)
time_start	time when the alarm starts
time_end	time when the alarm ends

IDs. To map p_connections to l_nodes this database table is necessary.

Table 3.6.: p_connections relevant fields

field name	description
id	unique identifier of the connection
default_node	mapping to the associated node (l_nodes)

Process Value Definitions: p_values In this database table possible process values to every machine are stored. Actually not the numerical value or time stamps, but the according connection, unit and a worded description.

Table 3.7.: p_values relevant fields

field name	description
id	unique identifier of the entry
connection	id of the associated connection (p_connection)
description_0	description of the process value

Process Values: p_values_archive This database table logs the process values with a specific time stamp and a numerical or optional an alphanumerical value.

ERP Database Tables

During the developing phase the ERP data source was not available. So the supervisors of this thesis demanded Audi workers to record the maintenance data manually in an Excel table. For the working schedule a

Table 3.8.: p_value_archive relevant fields

field name	description
id	unique identifier of the entry
value	id of the associated value (p_values)
value_num	numerical value
value_str	alphanumerical value
name_0	description of the entry
time_stamp	time stamp when the value was logged

workaround has been implemented. The excel file will be imported to a SMADA table named ”maintenance_actions” and the shift calendar will be generated to a SMADA table named ”shift_calendar”.

Incidents (Target Values): incidents_all Every maintenance action completed to any machine of the production line is logged to this database table. The entries of this table are also the target values of the supervised learning algorithm.

Table 3.9.: maintenance_actions relevant fields

field name	description
id	unique identifier of the entry
node	id of the associated end node (l_nodes)
time_stamp_start	timestamp when the maintenance starts
time_stamp_end	timestamp when the maintenance is ends
short_comment	shortcut describes a failed part
error_code	classifier for the maintenance action

Working Schedule: shift_calendar See subsection 3.3.1.

Table 3.10.: shift_calendar relevant fields

field name	description
id	unique identifier of the entry
node	id of the associated end node (l_nodes)
state	if of the machine state (mda_states)
time_stamp_start	timestamp when the state starts
time_stamp_end	timestamp when the state is completed

The database tables al_classes and al_message_archived, which are included in the data collection, are not relevant.

3.3. Feature Extraction

The whole data collection can be split up into machine related data. That means, every data record can be assigned to a specific node. The feature extraction process handles every node individually. In the prototype a list of nodes and their according IDs, which should be considered, needs to be chosen:

```
NODE_IDS = [2189, 2190, 2191, 2192, 2193, 2194, 2195]
```

During development of the prototype, data of two time windows was provided as a development data collection. It covers round about data from 01.04.2016 - 16.09.2016 and 13.09.2017 - 03.12.2017. Overall about 6 months of data. Every time window needs to be declared in the configuration file:

```
FEATURE_EXTRACTION_TIME_WINDOW_START = [
datetime(2016, 4, 1, 0, 0, 0, 0, tzinfo=pytz.timezone('UTC')),
datetime(2017, 9, 13, 0, 0, 0, 0, tzinfo=pytz.timezone('UTC'))]
FEATURE_EXTRACTION_TIME_WINDOW_END = [
datetime(2016, 9, 16, 0, 0, 0, 0, tzinfo=pytz.timezone('UTC')),
datetime(2017, 12, 3, 0, 0, 0, 0, tzinfo=pytz.timezone('UTC'))]
```

3.3.1. Shift Calendar

The ODC database collects data of machine state changes, like described in section 3.2. A single machine has about 30 000 state changes in a 6 months time frame. Many of the recorded states are low on information, because of a very small duration. To shrink this big amount of state changes an algorithm

Shift Calendar Node 2195

Start	Ende	Duration	Status
2016-03-01T15:51:23Z	2016-03-05T17:50:00Z	4 days, 1:58:37	1
2016-03-05T17:50:00Z	2016-03-06T23:58:28Z	1 day, 6:08:28	0
2016-03-06T23:58:28Z	2016-03-12T17:50:00Z	5 days, 17:51:32	1
2016-03-12T17:50:00Z	2016-03-13T05:50:00Z	12:00:00	0
2016-03-13T05:50:00Z	2016-03-13T17:50:00Z	12:00:00	1
2016-03-13T17:50:00Z	2016-03-15T05:50:00Z	1 day, 12:00:00	0
2016-03-15T05:50:00Z	2016-03-19T17:50:00Z	4 days, 12:00:00	1
2016-03-19T17:50:00Z	2016-03-20T23:30:27Z	1 day, 5:40:27	0
2016-03-20T23:30:27Z	2016-03-26T17:50:00Z	5 days, 18:19:33	1
2016-03-26T17:50:00Z	2016-03-29T08:04:46Z	2 days, 14:14:46	0
2016-03-29T08:04:46Z	2016-04-01T05:50:00Z	2 days, 21:45:14	1
2016-04-01T05:50:00Z	2016-04-02T07:33:59Z	1 day, 1:43:59	0
2016-04-02T07:33:59Z	2016-04-02T17:50:00Z	10:16:01	1
2016-04-02T17:50:00Z	2016-04-03T07:17:03Z	13:27:03	0
2016-04-03T07:17:03Z	2016-04-04T15:19:27Z	1 day, 8:02:24	1
2016-04-04T15:19:27Z	2016-04-04T22:33:40Z	7:14:13	2
2016-04-04T22:33:40Z	2016-04-09T17:50:00Z	4 days, 19:16:20	1
2016-04-09T17:50:00Z	2016-04-10T05:50:00Z	12:00:00	0
2016-04-10T05:50:00Z	2016-04-10T12:58:38Z	7:08:38	1
2016-04-10T12:58:38Z	2016-04-10T17:50:00Z	4:51:22	2
2016-04-10T17:50:00Z	2016-04-11T05:50:00Z	12:00:00	0
2016-04-11T05:50:00Z	2016-04-15T09:00:00Z	4 days, 3:10:00	1
2016-04-15T09:00:00Z	2016-04-15T17:50:00Z	8:50:00	0
2016-04-15T17:50:00Z	2016-04-16T17:50:00Z	1 day, 0:00:00	1
2016-04-16T17:50:00Z	2016-04-17T18:43:52Z	1 day, 0:53:52	0
2016-04-17T18:43:52Z	2016-04-23T17:50:00Z	5 days, 23:06:08	1

Figure 3.6.: Shift calendar of node id 2195 and a 1.5 months time frame.

has been implemented. It reduces the possible states to one of these three: error, running and a not-running state. The resulting shift calendar is needed for visual evaluation purposes and mostly because it covers not documented errors. In the configuration file a minimum and maximum state duration (in seconds) has to be defined:

```
STATE_MIN_DURATION_SECONDS = 300   # 5 minutes
STATE_MAX_DURATION_SECONDS = 10886400   # 18 weeks
```

To extract a shift calendar from the mda_state_changes table, the implemented algorithm executes the following steps:

1. Create an empty list L
2. Iterate through all mda_state_changes and add the first element to L
3. Map the current state to one of the three shift calendar states as:
   - (0) NOT_RUNNING: mda state 24, 26 and 27
   - (1) RUNNING: all other states

- (2) ERROR: mda state 20 or 23

4. If the current state equals the last added element in L continue with the next state at step 3
5. Calculate the state duration and if it's shorter than minimum state duration and continue with the next state at step 3
6. Update the ending time stamp of the last element in L to the current state starting time stamp
7. Add the current state to L

The resulting shift calendar can be viewed from within the prototype. An example (screenshot) is added to Figure 3.6.

3.3.2. Observations

Observation creation means to extract features in a predefined **2 hour** time window for every machine in the data collection. The resulting records are stored in the SMADA training_data database table:

Observations: training_data This database table covers two hour observation and extracted features as illustrated in Figure 2.5. There are a 4000 bytes placeholder field for comma-separated numerical observation features.

Table 3.11.: training_data fields

field name	description
id	unique identifier of the entry
node	id of the associated end node (l_nodes)
time	starting time of the two hours observation
attributes	placeholder for 4000 bytes comma-separated features
iterator	sequential number for each node
rul	time in hours till the next error will occur
target	compliant RUL class to the rul field

Spalten | Daten | Model | Constraints | Zugriffsberechtigungen | Statistiken | Trigger | Flashback | Abhängigkeiten | Details | Partitionen | Indizes | SQL

Sortieren... Filter:

	ID	DESCRIPTION_0
1	0	Anzahl der im Gateway ankommenden Meldetelegramme
2	1	Aus=1, Ein=0
3	2	Gesamtstückzahl
4	3	Gesamtstückzahl Typ 01
5	4	Gesamtstückzahl Typ 02
6	5	Gesamtstückzahl Typ 03
7	6	Gesamtstückzahl Typ 04
8	7	Gesamtstückzahl Typ 05
9	8	Gesamtstückzahl Typ 06
10	9	Gesamtstückzahl Typ 07
11	10	Gesamtstückzahl Typ 08
12	11	Gesamtstückzahl Typ 09

Figure 3.7.: Example process values of overall 73.

Spalten | Daten | Model | Constraints | Zugriffsberechtigungen | Statistiken | Trigger | Flashback | Abhängigkeiten | Details | Partitionen | Indizes | SQL

Sortieren... Filter:

	ID	EXT_CODE
1	0	2013
2	1	2014
3	2	2015
4	3	2016
5	4	2017
6	5	2018
7	6	2019
8	7	10299
9	8	10621
10	9	10655
11	10	14015
12	11	14500

Figure 3.8.: Example alarm codes of overall 573.

Observation features

Features for observations are extracted from the database tables al_archive (Table 3.5)and p_value_archive (Table 3.8). The prototype collects all process values and alarms occurring in an observation and calculates a feature by count, mean, difference or maximum value (section 2.3). In general there are two groups of features: alarm codes and process values. Because of the big amount of possibles different alarms and process values only the most frequently occurring ones are selected as features. These are 73 process values and 573 alarm codes. Overall a manual selected combination of count, mean, difference, minimum or maximum values are calculated, so that finally there are 1270 features per observation. An example of used process and alarm code values is given in Figure 3.7 and Figure 3.8. The screenshots are taken

from the database GUI Oracle SQL Developer[1].

Setting target values

RUL to classes The numerical value in hours, which describes the time to failure, has to be converted in to a class. The prototype predefines the number of 7 classes, but RUL time frames in ticks can be set in the configuration file. A tick is an observation iterator, which means one tick covers 2 hours:

```
TARGET_CLASS_RUL = [12, 24, 42, 66, 108, 150, 300]
```

RUL classes are set initially like stated in the example Table 2.1.

Incidents to RUL classes The following algorithm assigns target values to observation:

1. First a subset of incidents can be defined. Choosing only a specific subset of incidents for target setting may have great influence in the prototype performance. In the configuration file incident filter are handled by defining a starting string for the ”short_comment” field, of the incidents_all database table. Initially the filter is set to ”sp-”, which indicated a ”Spindelfehler” (en: spindle failure).

   ```
   TARGET_SHORT_COMMENT_STARTS_WITH = "sp-"
   ```

2. For each sample calculate the time difference, between the observation end time and the incident start time.

3. Assign RUL hours to the data record.

4. Calculate and assign the RUL class to the data record.

[1] http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html

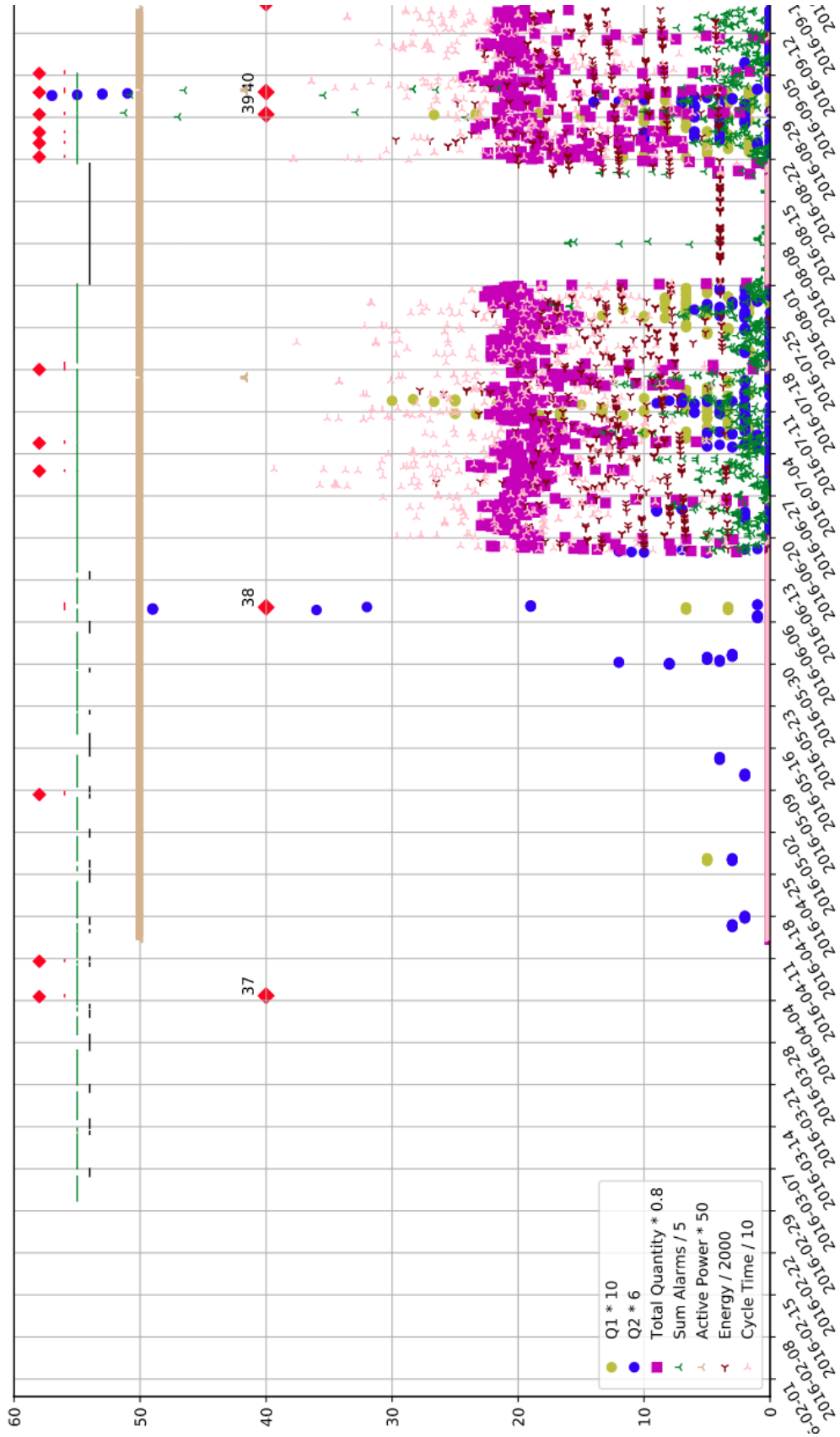

Figure 3.9.: smart data example

At this point, observation features, shift calendar and target values can be visualized from within the prototype. An example is given in Figure 3.9. The plot covers a selection of features in a scaled range. The red diamonds at y=40 show incidents from the database table. The diamonds at top represent machine errors from the shift calendar, also called undocumented failures. Between y=50 and y=60 the shift calendar is visible. A green line indicates running, red lines error and black lines that the machine is not running. The data shown in the plot is the visualization of some features.

3.3.3. Rolling Windows

As explained in section 2.4, somehow previous episodes of data should be added to an observation. To handle this requirement, a set of 6 rolling windows ($w_0, w_1, w_2, w_3, w_4, w_5$) is defined. One window aggregates features of all observations included. The window sizes, can be defined in the configuration file:

```
ROLLING_WINDOW_SIZES = [12, 24, 42, 66, 108, 150, 300]
```

Taken all together, rollings windows of an observation will represent input vectors as defined in Equation 2.11 and are stored in the training_data_extended database table (Table 3.12). For each observation feature, rolling window features are transformed by:

1. Calculating the mean value of all observation features included
2. Take the maximum value of all observation features included.

Overall, full samples will consist of 15240 features: 6 rolling windows * 1270 observation features * 2 (mean/max). The other database fields, RUL, time, target and iterator are adopted from the according observation of the training_data table.

Table 3.12.: training_data_extended fields

field name	description
id	unique identifier of the entry
node	id of the associated end node (l_nodes)
time	starting time of the two hours observation
x0_attributes	placeholder for the mean value of aggregated features
x0_attribute_max	placeholder for the maximum value of aggregated features
x1_attributes	placeholder for the mean value of aggregated features
x1_attributes_max	placeholder for the maximum value of aggregated features
x2_attributes	placeholder for the mean value of aggregated features
x2_attributes_max	placeholder for the maximum value of aggregated features
x3_attributes	placeholder for the mean value of aggregated features
x3_attributes_max	placeholder for the maximum value of aggregated features
x4_attributes	placeholder for the mean value of aggregated features
x4_attributes_max	placeholder for the maximum value of aggregated features
x5_attributes	placeholder for the mean value of aggregated features
x5_attributes_max	placeholder for the maximum value of aggregated features
iterator	sequential number for each node
rul	time in hours till the next error will occur
target	compliant RUL class to the rul field

3.4. Feature Selection

Limiting the amount of features, used either for training and testing, can be useful. Too many feature may add noise and have a negative influence to the prediction performance.

Like mentioned in section 2.7, this work achieves feature selection by ranking the random forest feature importance (Gini index). The measured feature rank, strongly depends on the training set and also on the target values (incidents). Here is an example on how features are ranked, by the Gini coefficient. In brackets the rating (in percent) is given:

```
1. feature 13079 (0.006916)
2. feature 5058 (0.006701)
3. feature 6336 (0.006647)
4. feature 13953 (0.005925)
5. feature 1246 (0.005887)
6. feature 10136 (0.004985)
```

```
7. feature 10143 (0.004544)
8. feature 8867 (0.004466)
9. feature 8866 (0.004453)
10. feature 6335 (0.004106)
11. feature 6287 (0.003975)
12. feature 2520 (0.003927)
13. feature 2537 (0.003871)
14. feature 3788 (0.003795)
15. feature 6326 (0.003769)
...
```

The number of rolling feature windows features, which will used for training an prediction can be defined as 15240. In practice this number differs, since features with no variance are removed before the actual feature selection (about 4000 remaining). Also the prototype selects only features which Gini coefficient is above the mean value of the preselected features. After the selection about 400 features are remaining.

3.5. Random Forest Modeling

When modeling a predictor, training sets are generated by choosing a subset of incidents in the configuration file:

```
INCIDENT_IDS_FOR_TRAINING = [8, 9, 11, 12, 13, 14, 15, 17, 18]
```

For each incident, training samples are by this algorithm:

1. Create an empty list L for training samples
2. Iterate through every incident
3. For each incident take the sample s closest to the incident by comparing the time stamps
4. Add each sample t which has a smaller iterator value than s to L if:

- To ensure that t is a complete sample (covering all rolling windows), the iterator should exceed a specific value, depending on the rolling window sizes.
- Only every third sample is considered for training. A sample is only added if t.iterator % 3 == 0.

The used classifier, random forest, provides several parameters to control the machine learning algorithm. Due experimenting the most important ones has been filtered and added as variables to the configuration file:

```
RANDOM_FOREST_NUM_TREES = 100
RANDOM_FOREST_MIN_SAMPLES_LEAF = 1
```

The first variable defines the number of estimators (trees) in the forest, which will vote for a class when predicting. The second parameter assigns a number on how many samples are required to split an internal node.

3.6. Ensemble Prediction

In the prototype, a prediction of a sample is made by calculating the true mean votes for a class of previous samples. The number of ticks to put into one ensemble can be defined in the configuration file:

```
PREV_TICK_DIFF_ENSEMBLE_PREDICTION = 120
```

To visualize the result of every single vote in a time series, an histogram is plotted. Unexpected behavior of the model can be figured out by this visualization. Especially when, making prediction for the smallest class, every position of the ensemble should be equally distributed over the histogram. Divergent distributions may indicated an defective model. The histogram x-axis represents the positions in the ensemble, and the y-axis the number of true votes. An example

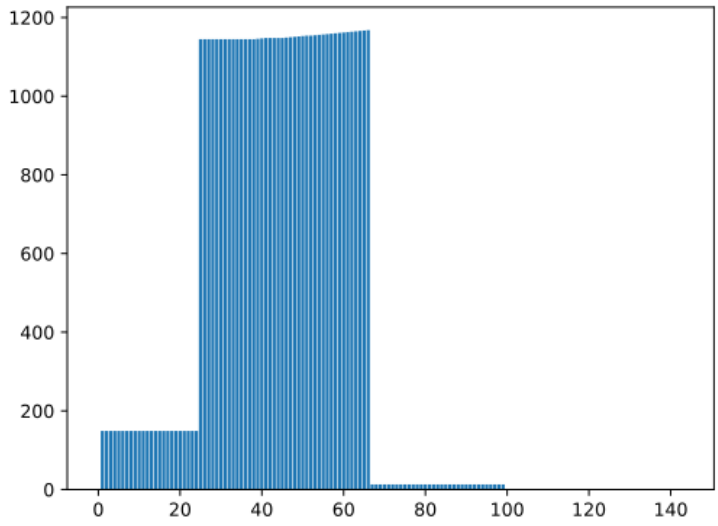

Figure 3.10.: Example: ensemble prediction histogram - 144 voters

is given in Figure 3.10.

The prototype provides a histogram for visualizing performance of the predictor for decision-making. The classes histogram Figure 3.11 shows the distribution of true classes in the ensemble.

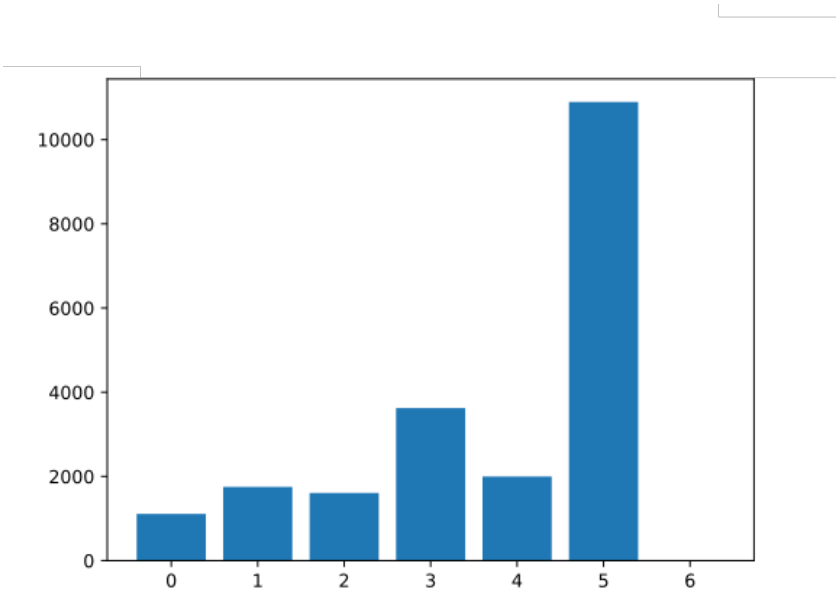

Figure 3.11.: Example: histogram RUL classes for a RUL class 1 predictor.

3.7. Evaluation

Table 3.13.: Evaluation machines

Name	ID	Connection ID
AF_220_1	2189	2013
AF_220_2	**2190**	**2014**
AF_220_3	**2191**	**2015**
AF_220_4	2192	2016
AF_220_5	2193	2017
AF_220_6	2194	2018
AF_220_7	2195	2019

In this section experiments and their evaluations are described. The prototype has data from 7 machines available to work with. As shown in Table 3.13, machines are part one work sequences AF220 (de: AF = Arbeitsfolge). Whereas two machines are picked for being training data machines (bold in the table) and the others will be test data machines for evaluation. The basis of choosing the two training machines is the quality and the significance of training samples (data before an error). As shown in the theoretical, part the basis of model evaluation is to alter parameters and their combinations. A finding is the performance variation of the predictive model, when changing parameters. The resulting predictive curve, of the testing machines and the best performed model, will be added in subsection 3.7.2. The threshold t and time in ticks the prediction has to increase before a positive is evaluated can be set in the configuration file:

```
PREDICTION_THRESHOLD = 0.45
PREDICTION_WAITING_THRESHOLD_SINCE_LAST_EVALUATION = 12
```

3.7.1. Experiments

Because the amount of data available is limited, no validation will be done (i.e. k-fold). The results should offer an idea on how well errors can be found by

the model and which parameters could improve predictions for future work. Training data will consist of machine 2190 and 2191 incidents. Testing machines are 2189, 2192, 2193, 2194, 2195. The class for all evaluations is class 3 with a RUL of 84 hours. By combination of the parameters ET, RM, RT3, EW and FS experiments will be evaluated. The duration of a single experiment calculation (one parameter set, 5 machines, each 3000 samples) takes about 10 minutes of processing time in the development environment. Using some advanced technologies, like distributed computing, would greatly speed up the whole evaluation process. Also, it would be easier to process more, than the chosen parameters. This work though is limited by ordinary workstation's infrastructure (Figure A.1). In the features selection step, from originally 15240, 11220 features show no variance in the data and are dropped. From the remaining features, random forest selects about 300 features, which have the most relevance to the training set targets. To visualize experiment results, evaluation graphics are introduced. They are scatter plot showing the relation between true positives (x-axis) and the precision (y-axis, between 0 and 1). The visualization of different parameters (shape, border and color) are defined in the next section. By counting the errors (red diamonds in the SMADA graphics, see Figure 3.9) of testing machines, a maximum of 92 errors is assumed.

Parameter Set

The evaluation starts with the following parameter variation:

1. Feature Selection (FS):
    a) Feature Subset 1: Rank features by Gini
    b) Feature Subset 2: All features
2. Ensemble Window (EW):
    a) Ensemble Window 1: Size: 120, all predictors
    b) Ensemble Window 2: Size: 120, every 5^{th}
    c) Ensemble Window 3: Size: 120, last 30, all predictors

   d) Ensemble Window 4: Size: 120, first 30, all predictors
   e) Ensemble Window 5: Size: 120, second 30, all predictors
   f) Ensemble Window 6: Size: 120, third 30, all predictors
   g) Ensemble Window 7: Size: 1, no ensemble window, but probability prediction, see section 2.10

3. RF Num Trees (RT), in brackets the shapes of evaluation plot points:
   a) RF Trees 1: 100 (square)
   b) RF Trees 2: 200 (circle)
   c) RF Trees 3: 300 (diamond)
4. RF Min Samples (RM), in brackets the border color of evaluation plot points:
   a) RF Min Samples 1: 1 (black)
   b) RF Min Samples 2: 2 (blue)
   c) RF Min Samples 3: 4 (green)
   d) RF Min Samples 4: 8 (red)
5. Evaluation Threshold (ET), in brackets the color of evaluation plot points:
   a) Threshold 1: 30 % (red)
   b) Threshold 2: 40 % (orange)
   c) Threshold 3: 50 % (yellow)
   d) Threshold 4: 60 % (green)

Experiment 1: FS1 + EW1 + RT* + RM* + ET* (48 single evaluations) The first experiment shows a result below expectation, by knowledge of results during prototype development. The attempt of limiting the features to better the prediction performance gone wrong. As visible from the evaluation plot in Figure 3.12 the precision in average is about 60%, but is expected to be about 65%. So from now on, all of about 4000 features showing variance in the data will be chosen.

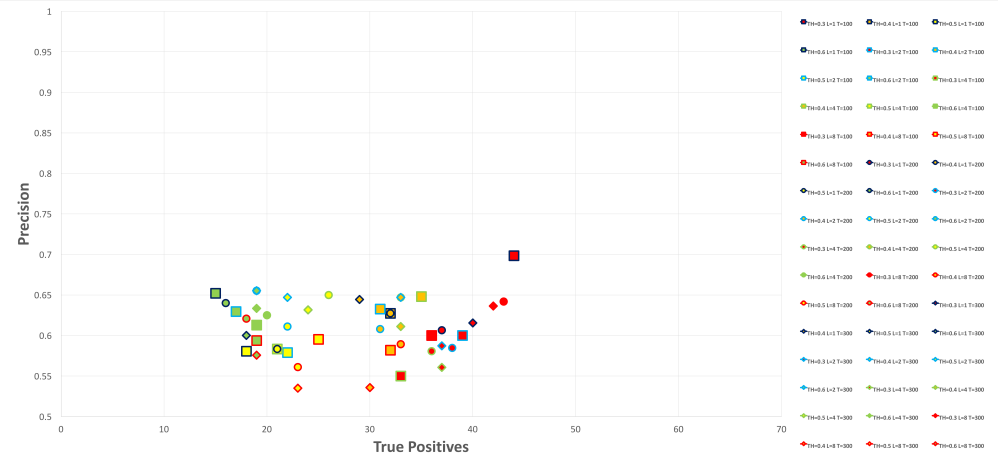

Figure 3.12.: Experiment 1: FS1 + EW1 + RT* + RM* + ET* (48 single evaluations)

Experiment 2: FS2 + EW1 + RT* + RM* + ET* (48 single evaluations) This experiment equals the first one, but no features are eliminated for training. Even with about 10 times more features the machine learning algorithm shows no significant change in calculation time. So the assumption, of using many features has a strong influence to the computational performance is wrong for random forest, but should hold for other machine learning algorithms. Looking at the evaluation plot (Figure 3.13) in comparison to the previous experiment, data points are shifted upwards by about 5%, which signals a higher precision, and a similar data range in the x-axis as in first experiment. Looking at the top left corner of the scatter plot it is noticeable, that there are almost green data points, which visualizes a prediction threshold of 60% (ET4). From left to right other thresholds become more frequent in order from 50% (ET3 yellow), 40% (ET2 orange) to 30% (ET1 red). As expected, a smaller threshold results in a lower precision, but a higher true positive hit rate. Surprisingly, shapes (diamond, circle and square), which indicates the number of trees in a forest, are equally distributed over the whole scatter plot. The numbers were chosen (100-300 trees) to recognize a slight improvement when using more trees, but from a visual point of view, when looking at the evaluation plots, one can't perceive a trend, regarding the shapes.

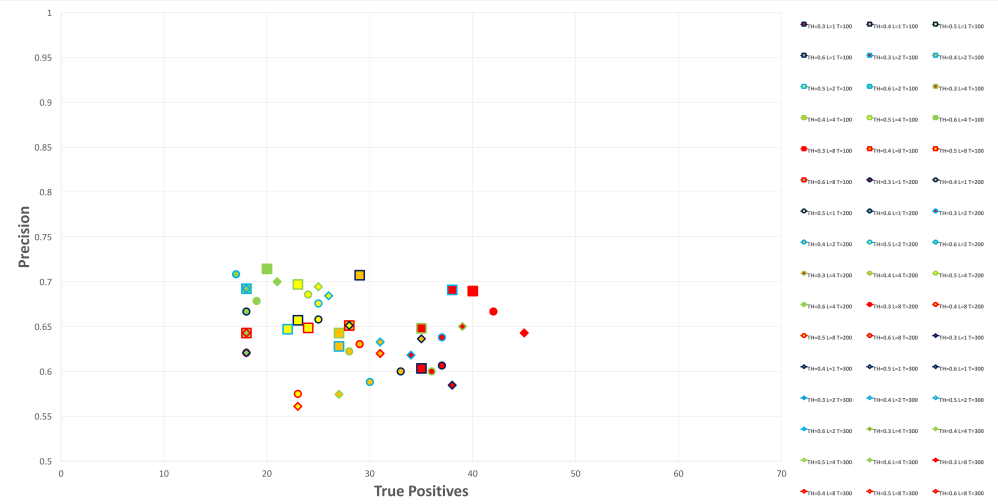

Figure 3.13.: Experiment 2: FS2 + EW1 + RT* + RM* + ET* (48 single evaluations)

Experiment 3: FS2 + EW2 + RT* + RM* + ET* (48 single evaluations) This experiment is the same as the previous one, except the ensemble prediction skips four of five voters, which reduces the amount of original 120 predictors to 120 / 5 = **25**. The goal of this test is to find out on how the ensemble prediction window design influences the prediction. With fewer voters the ensemble prediction is more likely to jump, caused by reduced resolution in possible prediction percentage (multiple of 5/120). The results are visible in Figure 3.14. It is conspicuous that the results are similar to the previous experiment, but the scatter points are stretched to right, by about the double x-axis width. The better evaluation results are still green at a 73% precision, but the true positive rate increased and sticks around 30. This shows, that the ensemble prediction design has influence on the prediction performance. This idea will be further evaluated in the next experiments.

Experiment 4: FS2 + EW3 + RT* + RM* + ET* (48 single evaluations) In this experiment the influence of the last quarter of the whole 120 predictor ensemble window is demonstrated. In comparison to the previous two experiments the average hit rate of true positives is at about 35, which can be seen as an improvement, by reason of equal precision (Figure 3.15). In comparison

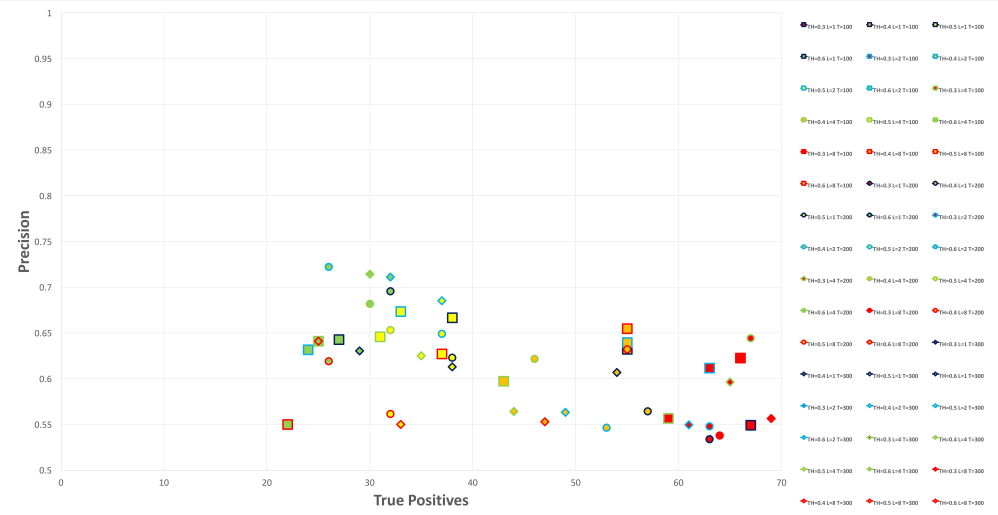

Figure 3.14.: Experiment 3: FS2 + EW2 + RT* + RM* + ET* (48 single evaluations)

to the second experiment, that means, there is maybe a group of successive predictors in the ensemble window, responsible for a worse result.

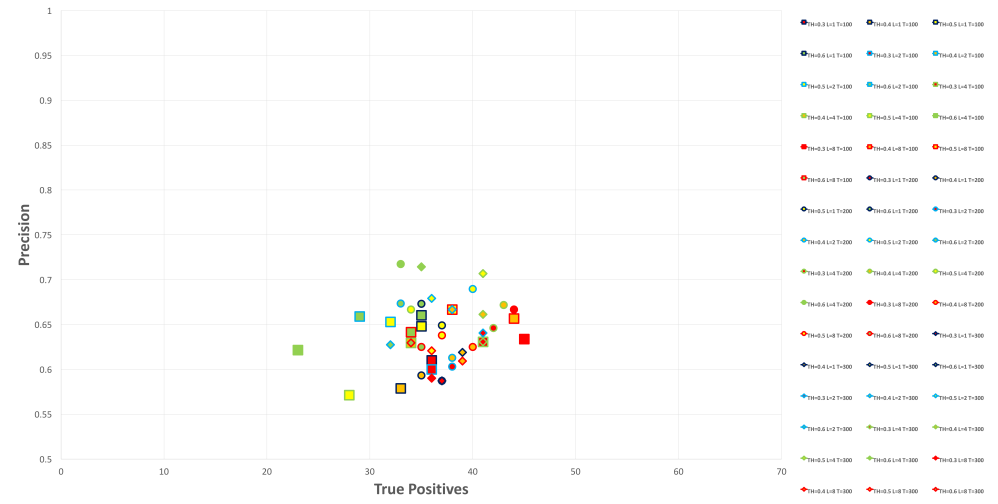

Figure 3.15.: Experiment 4: FS2 + EW3 + RT* + RM* + ET* (48 single evaluations)

Experiment 5: FS2 + EW4 + RT* + RM* + ET* (48 single evaluations) Based on the insight of the fourth experiment, the opposite approach will be tested. Instead of taking the last quarter of the ensemble windows, only the first quarter will be chosen. As visible in Figure 3.16 the results are much worse

than in the previous experiments. One can derive that, the predictors closer to the sample is influencing the prediction badly.

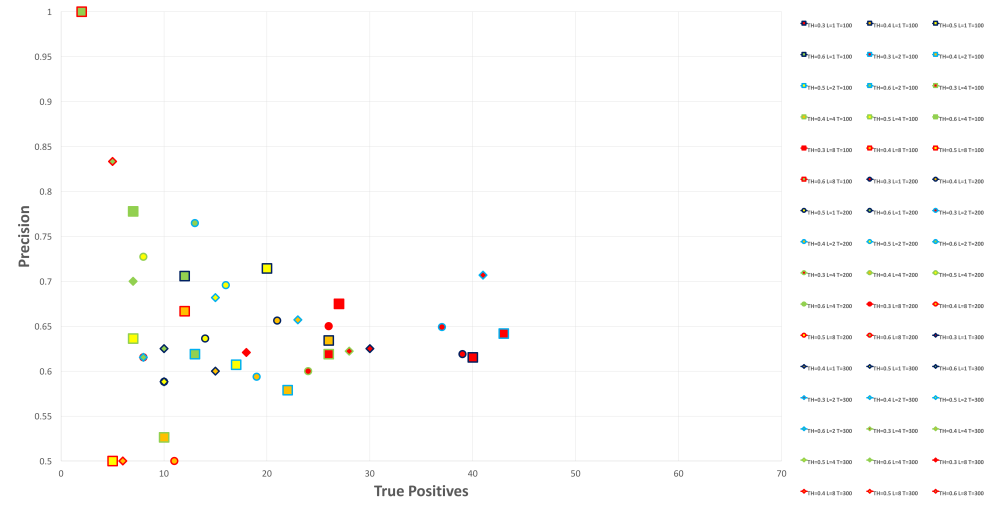

Figure 3.16.: Experiment 5: FS2 + EW4 + RT* + RM* + ET* (48 single evaluations)

Experiment 6 + 7: FS2 + EW5/EW6 + RT* + RM* + ET* (48 single evaluations) The sixth and seventh experiment have almost the same results. They chose either the second or the third quarter of the 120 predictor ensemble. As shown in Figure 3.17, the results are slightly below the fourth experiment, but better than the fifth one.

Experiment 8: FS2 + EW6 + RT* + RM* + ET* (48 single evaluations) This experiment shows the advantage of the ensemble prediction approach over a probability prediction. Only the first, which is the actual sample, is responsible for the prediction probability. Random forest is calculating the probability of the given class, and the value is directly taken for the prediction curve. The whole evaluation process, is based on the ensemble prediction approach, which means that the chosen metrics is not working for this experiment. The sample prediction is independent of the previous predictions, and therefore more likely to jump. As shown in Figure 3.18, due the unsteady prediction curve, one failure can be marked multiple times as a true positive. But overall

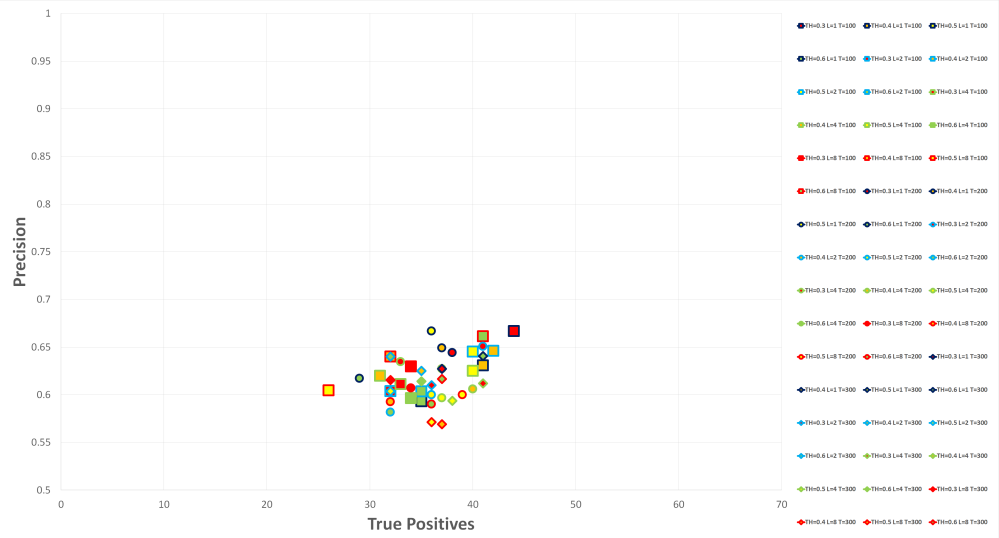

Figure 3.17.: Experiment 6: FS2 + EW4 + RT* + RM* + ET* (48 single evaluations)

the resulting curve is not as intuitive to interpret as the ensemble prediction curves shown in subsection 3.7.2.

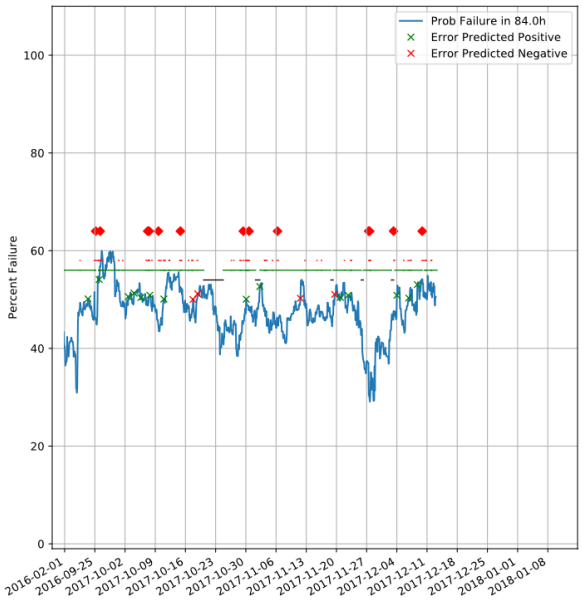

Figure 3.18.: Probability prediction - Evaluation not working

All experiments To visualize the whole experiment set, and to make them comparable, an according plot is shown in Figure 3.19. The yellow point in the black square is the experiment showing the result handled in the next section, because of its good relation between true positives and precision.

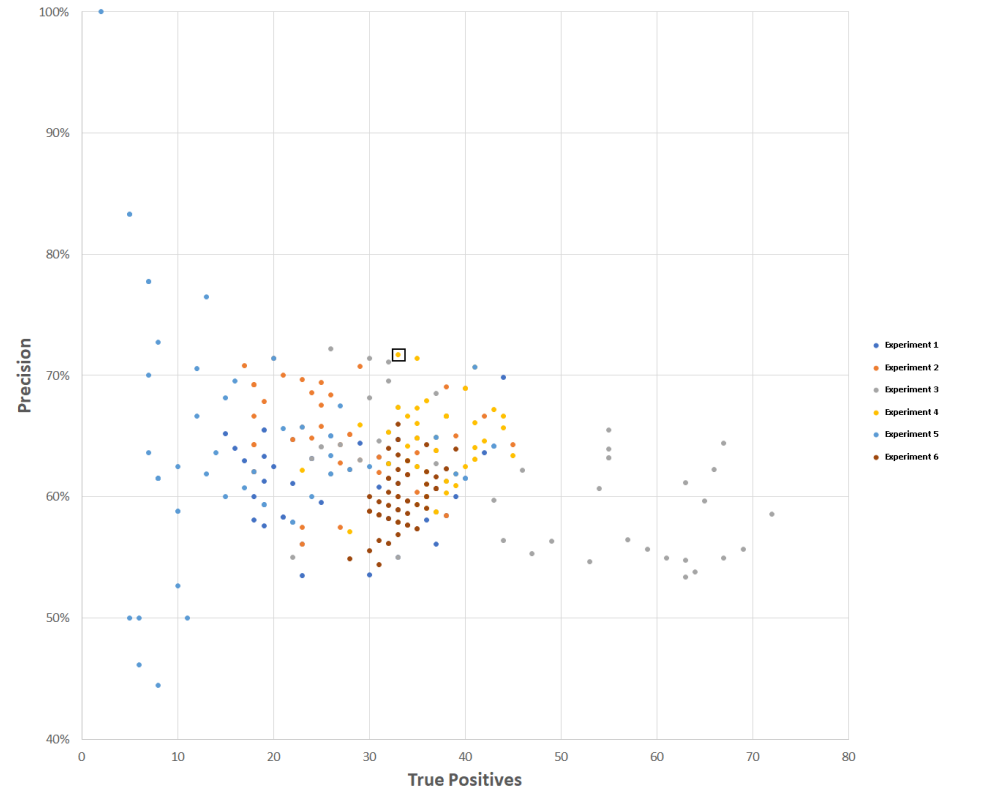

Figure 3.19.: All experiments

3.7.2. Results

Finding a suitable relation between true positives and precision is the challenging part in such data analysis task. Finding as many failures as possible in advance, can reduce costs in production, but many false positives can lead to premature maintenance actions, followed by higher costs. Especially in this task, one can assume that when the prediction curve exceeds the threshold, the machine is in a bad condition, but a detailed description of the error is missing. The result of this work, is the knowledge of, that failures can be recognized, by a condition monitoring prediction curve. Anyway, to derive concrete mainte-

nance action from it is, in this case, not possible. As the best model, one from the fourth experiment is chosen (Threshold: 0.6, Leafs: 4, Trees: 200), and the resulting prediction curves, for each testing machine, are added to the next section.

Interpretation The prediction curves covers predictions from two time frames (April – September 2016 and September – December 2017). The blue prediction curves on the y-axis describes the probability of a machine failure in 84 hours and the x-axis are continuous timestamps, whereas one grid on the x-axis covers one week. Red diamonds on the horizontal y = 40 line are documented failures. Red diamonds above the x = 60 line are undocumented failures, with a duration of minimum two hours, extracted from the mda_state_changes (Table 3.2) database table. The horizontal lines at y = 54, y = 56 and y = 58 are visualized machines states. Green means the machine is running, black the machine is not running and red means, that the machines is in an error state. In the well working prediction example shown in Figure 3.24, the prediction curve catches all three documented errors. Also, most of the times. when the prediction curve exceeds the threshold of 60% there is a significant occurrence of red diamonds. In the other hand Figure 3.20 illustrated the difficulty of handling undocumented errors in evaluation. Looking at the leftest false positives (red x), the prediction curves reaches the maximum of 100 percent about four days before an error occurs. The error though, has a duration of less than two hours, so it is not used for evaluation, since it is rated as a minor failure. The model shows a significant change in the data at that point, but the real hardness of the error is unknown. In the section the prediction curves of all testing machines are added. They should give a visual overview of the prediction performance.

3.7.3. Resulting prediction curves

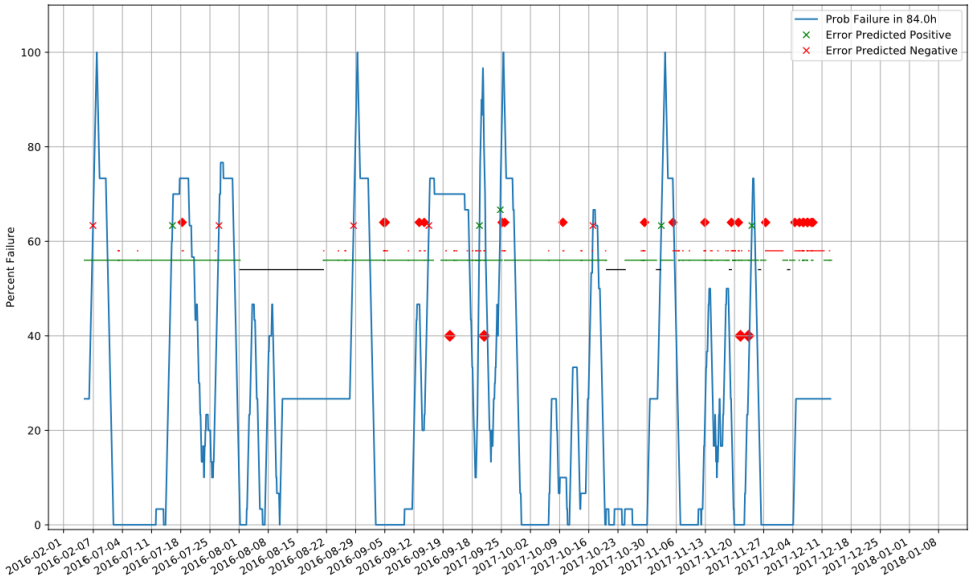

Figure 3.20.: Prediction Curve 2189

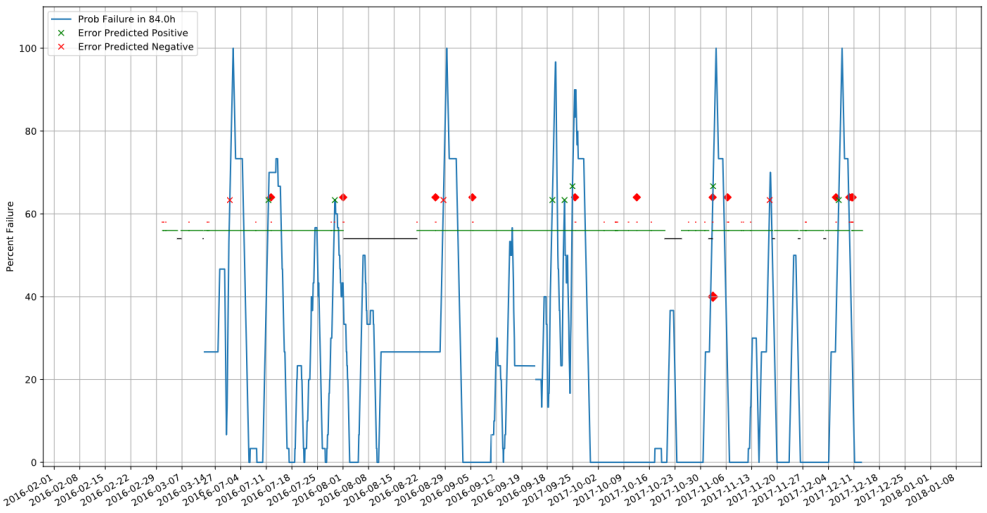

Figure 3.21.: Prediction Curve 2192

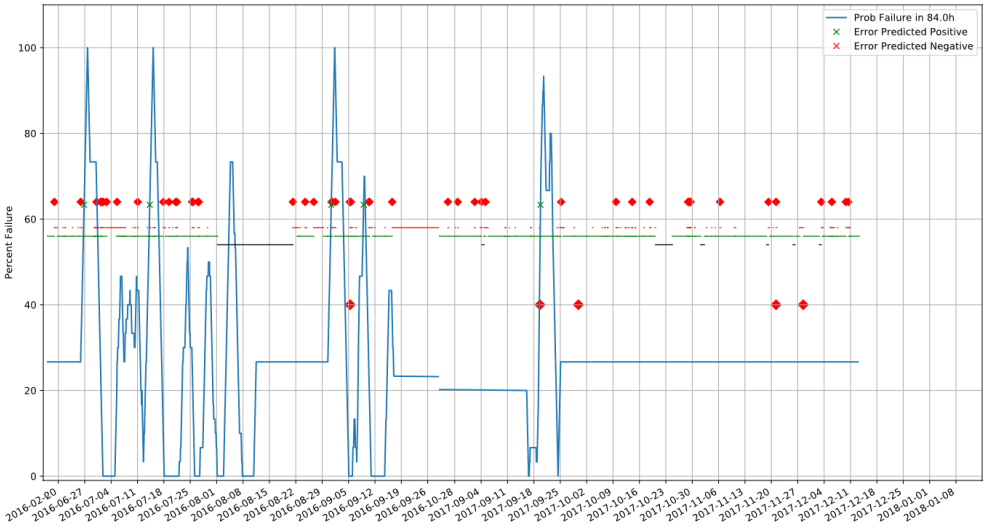

Figure 3.22.: Prediction Curve 2193

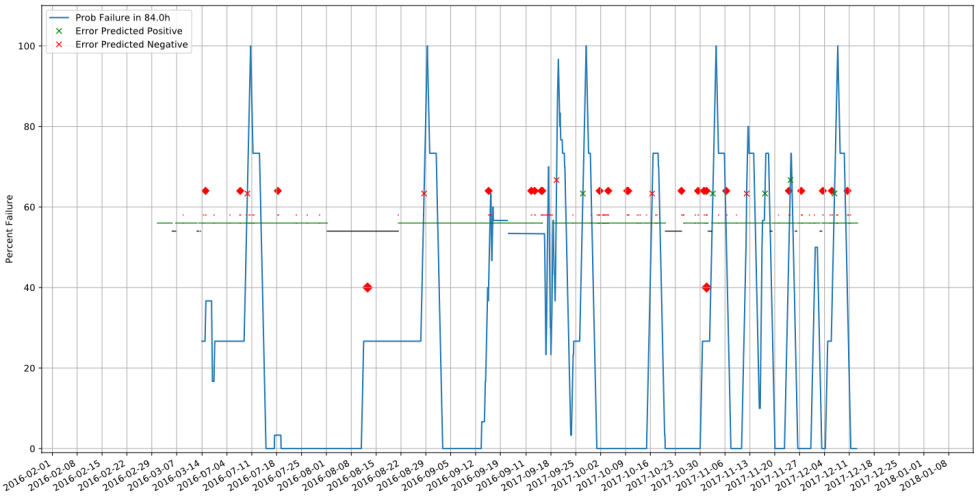

Figure 3.23.: Prediction Curve 2194

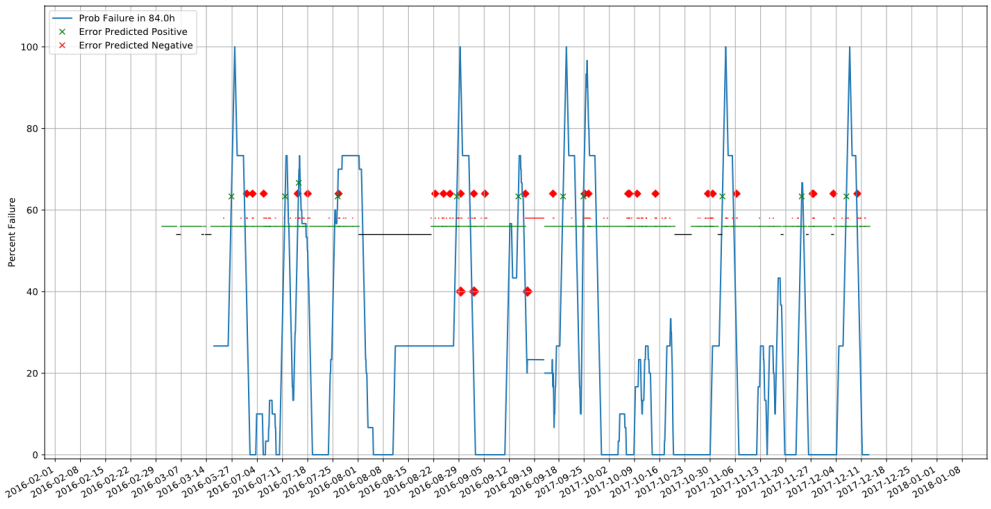

Figure 3.24.: Prediction Curve 2195

4. Conclusion

Lessons Learned Time series data analysis for an industrial environment is an uncertain task. Since there are no uniform solutions, different methods, ideas and strategies need to be tried. This can be very time-consuming when working with big data sets. Also, one may not know if a solution is the best possible one, and if there is room for improvement by applying different algorithms and methods. It is necessary to be creative and flexible to react on changing initial assumptions of the data and results occurring during work.

Is the available data suitable for predicting machine breakdowns? Yes. The data analysis process shown in this thesis covers all steps necessary, but partially in a trivial way. There may be room for improvement, but for now the results are promising. The hardest part of the analysis was to handle the lack of complete maintenance and failure documentation. A workaround has been implemented, but obviously it is not the best way to do it. Good target values are the basis of well-trained models and necessary for reliable supervised machine learning. It shows that predictive data analysis has to be strongly integrated into the organization and that Audi Hungaria may not be ready to include a predictive maintenance system in their environment yet. It is advantageous to start collecting and storing available data and complete maintenance documentation now, to benefit from actual data as future training data.

How can the implemented prototype can be improved for future work? The time series analysis approach shown in this thesis is just one way to do it. Not every aspect of the analysis process has been implemented in such detail. A refined feature engineering process, which finds the best possible features,

is missing. In this thesis from all features only the simple statistical values like mean, max, min and the count has been calculated. Strong features are essential for a powerful model and by unsupervised or supervised feature engineering the model performance can be improved. Another important aspect of data analysis is the visualization. In this work the results are simply plotted as percentage on how likely a machine is to fail at a specific time. In production more information could be given and visualized to the machine or maintenance worker by visual analytics systems [18]. The learning algorithm used in this work (Random Forest) is suitable for the given task, but trying out different methods may train better models. As already mentioned, complete machine failure documentation is essential and lead to more certainty about the data and in the end also a more stable predictive models.

Appendices

A. Technical Documentation

A.1. Introduction

This documentation gives a rough overview of the software source code and the development environment. It includes a major part of the thesis work. The prototype is based on the decision that one of the best tools to implement a data analysis solution is the Python programming language. There are many libraries to apply machine learning algorithms and also for result plotting. One requirement is that the prototype should work on any device, which almost limited the choice to a web framework, whose front-end will be accessible from within a web browser. Another requirement is, that the software should handle Oracle databases, which narrows down the choice to Django[1], a Python open source web server project. To simulate a productive environment (manufacturing plant), a prototype infrastructure at the institutes local network is set up. As shown in Figure A.1 it has the following components:

- **VM Host:** The prototype and database server, which will be provided by Audi, will be running on its own machine. In the development environment a VM client for each server is installed on a VM host machine.
- **VM Oracle database server:** This is the place where Audi would store the RAWDA database and were the prototype stores the smart data.
- **VM Django web server:** This server will be called by the web clients to access the prototypes' user interface.

[1] `https://www.djangoproject.com`

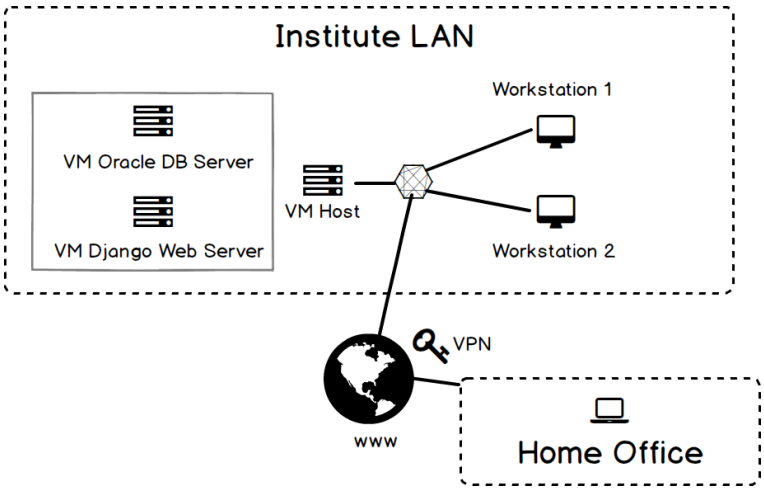

Figure A.1.: Prototype Development Environment

- **Workstations and Home Office:** The servers are accessible from within the institutes LAN and the universities network. That's why working on the prototype from home also over a VPN connection is possible. All workstations are using the free student licensed version of "Jetbrains PyCharm IDE"[2], which was an enormous help during the whole software development process.

A.2. Django Framework

A.2.1. Packages

The prototype is based on Django, but there are several other packages the software depends on:

- Django (v2.0.3): Python web framework
- cx_Oracle(v6.2.1): Database package for Oracle databases
- matplotlib(v2.2.2): Packages for drawing plots
- numpy(v1.14.2): Package for scientific computing (matrices)
- scikit-learn(v0.19.1): Library for machine learning algorithms

[2] https://www.jetbrains.com/pycharm/

- and others

A.2.2. Multiple Database Setup

In this project multiple databases are used. The two external databases SMADA and RAWDA, where the models and the data for the program logic are stored and also an internal database is needed (SQLlite) to store the Django-specific data. To setup this special situation the following steps are needed to be done:

1. Add the database configurations to settings.py:

```
DATABASES = {
    'default': {
    'ENGINE': 'django.db.backends.ite3',
    'NAME': os.path.join(PROJECT_DIR, 'project.db'),
  }, 'smada': {
    'ENGINE': 'django.db.backends.oracle',
    'NAME': 'orcl',
    'USER': '*',
    'PASSWORD': '*',
    'HOST': '*',
    'PORT': *
  }, 'rawda': {
    'ENGINE': 'django.db.backends.oracle',
    'NAME': 'orcl',
    'USER': '*',
    'PASSWORD': '*',
    'HOST': '*',
    'PORT': *
  }
}
```

   After that three databases with the labels ”default”, ”smada” and ”rawda” are added to the project.

2. Assign database to Django apps. Add database router to setting.py:

```
DATABASE_ROUTERS = ['project.router.Router']
```

3. Add the routing file to the project tree "./project/router.py":

```
class Router(object):
  def db_for_read(self, model, **hints):
    if model._meta.app_label == 'smada':
      return 'smada'
    elif model._meta.app_label == 'rawda':
      return 'rawda'
    return None

def db_for_write(self, model, **hints):
  if model._meta.app_label == 'smada':
    return 'smada'
  elif model._meta.app_label == 'rawda':
    return 'rawda'
  return None

def allow_relation(self, obj1, obj2, **hints):
  if obj1._meta.app_label == 'smada' or \
    obj2._meta.app_label == 'smada':
    return True
  elif obj1._meta.app_label == 'rawda' or \
    obj2._meta.app_label == 'rawda':
    return True
  return None

def allow_migrate(self, db, app_label, model_name=None, **hints)
    :
  if app_label is 'smada' or db is 'smada':
    return True
  else:
    return False
```

By the Django app name, the calling object is related to, the routing file decides, which database should be used for read, write, allow_relation and allow_migrate actions. If the database is not found, ”None” will be returned and the default database will be chosen.

A.3. Model and Database

A.3.1. Migrate Database

Model class definitions in ”models.py” files of Django apps will automatically be connected to the app related database by the Django ORM component. Initially or if changes to these files are made (i.e. data fields added) the underlying database needs to be updated. In Django this process is called ”migration”. Here is an example how to migrate the ”smada” database from the console:

```
cd project_root
Python manage.py makemigrations smada
Python manage.py migrate smada --database smada
```

A.3.2. SQL Query Performance Boost

Dealing with many datasets can lead to performance issues, so queries need to be executed carefully and may be optimized by:

Temp Tabless The process values database of the development data has about 80 000 000 datasets. Each assigned to exact one end node. When processing this data it may sufficient to make queries only on the data of that end node, instead of using all rows. So first copy the relevant values to a temp table, and then make the queries from there. To copy the process values of a specific node to the temp table the following SQL query is used in the source code:

```
INSERT INTO AL_ARCHIVE_TEMP (ID, EXT_CODE, TIME_START, TIME_END)
     SELECT AL_ARCHIVE.ID, AL_MESSAGES.EXT_CODE, AL_ARCHIVE.
    TIME_START, AL_ARCHIVE.TIME_END FROM AL_ARCHIVE INNER JOIN
    AL_MESSAGES ON AL_ARCHIVE.MESSAGE = AL_MESSAGES.
    MESSAGE_ARCHIVED WHERE AL_ARCHIVE.NODE = 2189
```

The temp table needs to be created first. It also has to be truncated, immediately before copying new data into it.

Defer and Only When using the Django ORM implementation, every data field of the record is populated to the Python object when accessing them. This could slow the performance down especially when using ”varchar”. It can be avoided by ”defer()” and ”only()” methods in a QuerySet. The ”defer” command avoids the given fields to be queried, as the ”only” command will limit the populated fields to the specified one.

For example the following alarm object only has the ”ext_code” field immediately accessible:

```
alarm = AlArchive.objects.get(id=5).only("ext_code")
```

,whereas this alarm object has all data fields, except ”ext_code”, assigned:

```
alarm = AlArchive.objects.get(id=5).defer("ext_code")
```

When trying to access not assigned fields, they will be lazy loaded, so it is working, but may take some additional time to execute the according SQL query.

A.4. Project Folder Structure and Files

In this section the project's root folder structure, included files and sub folders are described.

A.4.1. DJANGO Backend - Root Folder

Folder: project

The projects folder contains Django configuration files:

1. **router.py**: Handles routes to assign apps to different databases.
2. **settings.py**: Main Django configuration file. For example, database connections are defined here.
3. **urls.py**: Routes web client requests to the according function

Folder: rawda

The rawda folder is an Django app which covers functionality to handle the rawda database

1. **models.py**: In this file the rawda ORM definitions are added.
2. **logic.py**: Helper functions for working with the rawda database.

Folder: smada

The smada folder is an Django app which covers functionality to handle the smada database. In general it transforms and stores raw data to the smada database.

1. **config.py**: Main data analysis configuration file as outlined in the practical part.
2. **logic.py**: Helper functions for smada record generation.
3. **models.py**: SMADA ORM definitions.

4. **data_selection.py**: Shift calendar generation.
5. **data_extraction.py**: Feature extraction process.
6. **data_training.py**: Predictive model creation.
7. **data_prediction.py**: Prediction process.
8. **data_evaluation.py**: Evaluation workflow.

Folder: web

The web folder is an Django app which covers functionality to handle web client request. Basically all the communications between server and client is done by AJAX-request. The returned JSON-files are handled by the frontend AngularJS app.

1. **api.py**: All possible web client request are implemented in this file.

A.4.2. Angular Frontend - web/static/*

The frontend is a static single page website, based on the AngularJS JavaScript framework [3].

1. **js_app.js**: AngularJS controller and routing.
2. **pages/*.html**: Template files for dynamically generated AngularJS pages.
3. **images/*.svg + images/*.png**: Plots generated from the backend.

A.5. User Interface

The prototype has two screens available from where the user can take actions. First the main screen, where a user can execute different actions for every node at once and secondly a node screen, where a user can take node specific actions.

[3] https://angularjs.org

A.5.1. Main Screen

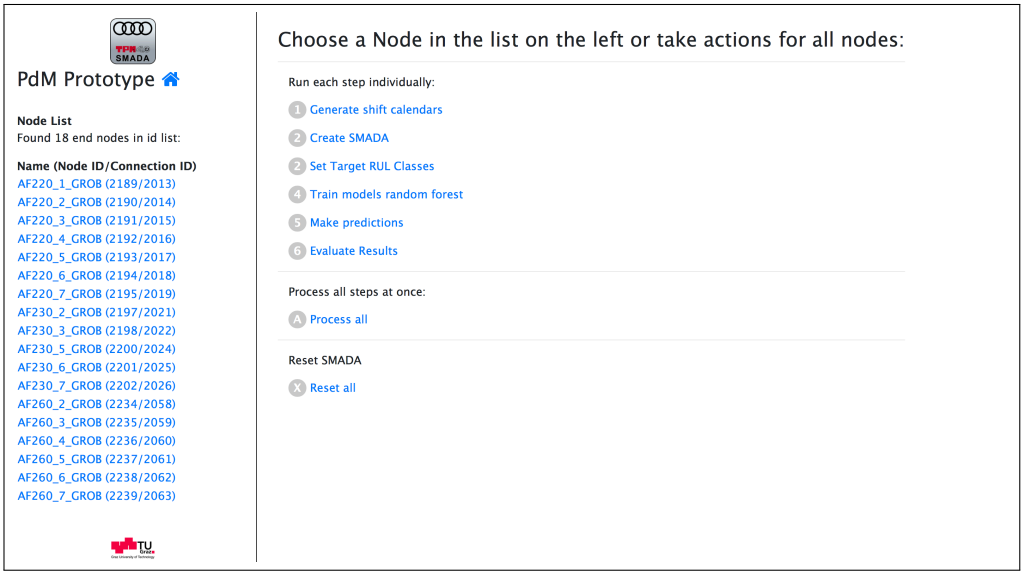

Figure A.2.: Prototype screenshot: main screen

From the main screen the following actions can be started:

1. Generating shift calendar for all nodes.
2. Creating smada for all nodes.
3. Setting target values as defined in the configuration file. When changing target RUL values it is not necessary to go through the whole feature extraction process (smada creation) again. It can be done separately.
4. Training Models: After changing random forest model parameters, the model needs to be retrained by this action.

A.5.2. Node Screen

This screen offers an interface to take actions on a single node:

1. Generating shift calendar.
2. Creating smart data.
3. Making prediction for all RUL classes.

4. Displaying the shift calendar.

5. Plotting the smart data.

6. Showing the prediction curve for all RUL classes.

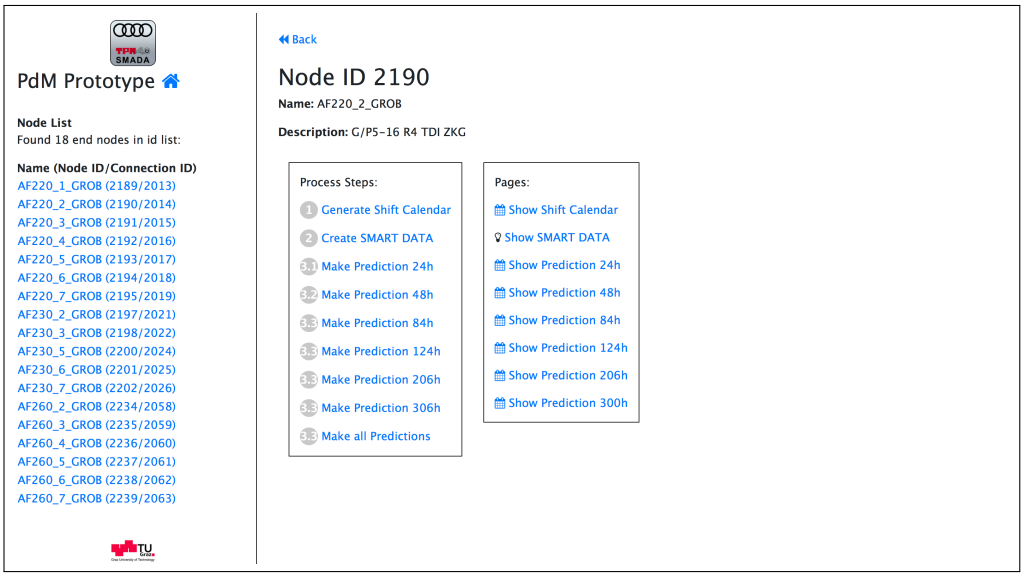

Figure A.3.: Prototype screenshot: node screen

Bibliography

[1] J Wang et al. *Predictive maintenance based on event-log analysis: A case study*. 2017.

[2] Ruben Sipos et al. “Log-based predictive maintenance”. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’14*. 2014, pp. 1867–1876.

[3] V. Ojanen. “Maintenance innovations - Types, patterns and emerging trends”. In: *ICMIT 2014 - 2014 IEEE International Conference on Management of Innovation and Technology*. 2014, pp. 321–326.

[4] B K N Rao. *Handbook of Condition Monitoring*. 1996, pp. 37–48.

[5] Johannes Helbig Henning Kargermann, Wolfgang Wahlster. “Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0”. In: *Bmbf.De* April (2013), pp. 1–116.

[6] Doug Laney. “3D data management: Controlling data volume, velocity and variety.” In: *META Group Research Note* 6.February 2001 (2001), p. 70.

[7] Tobias Meudt, Malte Pohl, and Joachim Metternich. *Modelle und Strategien zur Einführung des Computer Integrated Manufacturing (CIM) – Ein Literaturüberblick*. Tech. rep. 2017, p. 27.

[8] A.-W. Scheer. *CIM - Der computergesteuerte Industriebetrieb*. 1988.

[9] Bilal Esmael et al. “A Statistical Feature-Based Approach for Operations Recognition in Drilling Time Series”. In: 5 (2013), pp. 454–461.

[10] L Breiman et al. *Classification and Regression Trees*. Vol. 19. 1984, p. 368.

[11] Tan Pang-Ning, Michael Steinbach, and Vipin Kumar. *Introduction to data mining*. 2006, p. 796.

[12] Leo Breiman. “Random forest”. In: *Machine Learning* 45.5 (1999), pp. 1–35.

[13] Sebastian Raschka. *Python Machine Learning Unlock*. Vol. 22. 2. 2015, pp. 137–141.

[14] Ron Kohavi. “A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection”. In: *Appears in the International Joint Conference on Articial Intelligence (IJCAI)*. 1995, pp. 1–7.

[15] M. Leutbecher and T. N. Palmer. “Ensemble forecasting”. In: *Journal of Computational Physics* 227.7 (2008), pp. 3515–3539.

[16] J D Hamilton. *Time Series Analysis*. 1994.

[17] Johannes Pan. “Change Management in Maintenance - Feasibility Study: From a Decentral Controlled Maintenance Organization to a Central Managed Maintenance Structure”. In: *Master Thesis* (2018).

[18] James J. Thomas and Kristin A. Cook. “Illuminating the path: The research and development agenda for visual analytics”. In: *IEEE Computer Society* (2005), p. 184.