Improving Cargo Train Availability with Predictive Maintenance: An Overview and Prototype Implementation

Sebastian Kauschke

kauschke@tk.tu-darmstadt.de
Knowledge Engineering Group & Telecooperation Group
Technische Universität Darmstadt, Germany

Abstract. In cargo transportation, reliability is a crucial issue. In the case of railway traffic, the consequences of locomotive failure are not limited to the affected machine. Beside the cost of the machine itself, delays are caused and ultimately propagated through the railway network, rendering the accumulated cost of a single incident unpredictable. In order to avoid failures, *Predictive Maintenance* (PM) can be used. Predictive Maintenance targets the substitution of existing maintenance processes (e.g. time-based preventive maintenance) by conveniently scheduled corrective maintenance through exploitation of the underlying deterioration processes. In an ideal PM scenario, constant monitoring of the machine is available, measuring all relevant variables, e.g., temperatures or vibrations on a regular basis. However, in the real world, this assumption is limited: The hardware often does not deliver the required amount of data in the necessary precision. Often the machines record only a log-file which provides all activities – useful or not – that the various systems in the machine keep track of.
In this paper, we give a short overview on PM in general and on the various types of systems that can be considered for PM. We elaborate on the differences in data as well as the nature of the systems it is possible to predict failures upon.
In a prototypical example, we make use of machine learning methods to construct a failure prediction model for cargo trains. This data-driven approach focuses on a specific failure problem which is important to improve upon and aims at an easy prototype implementation for the currently available system.
We train a classification model which uses the pattern structure of the diagnostic-messages of the locomotive to recognize abnormal activities in the locomotives behaviour. A meta-classification layer on top of this anomaly detection allows us to build a prediction mechanism. We evaluate our findings on the data of 340 locomotive tours and elaborate on possible improvements of the method.

1 Introduction

Predictive maintenance (PM) scenarios usually evolve around big machinery. This is mainly caused by those machines being both expensive and important for production processes of the company they are used in. Failure of these machines usually have a plethora of negative effects, some of them causing chain reactions that affect further process steps. A successful predictive maintenance method for a machine can help at

preventing this, aid in planning for resources and material, and reduce maintenance cost and production downtime. In order to benefit from PM, a constant monitoring and recording of the machine status data is required.

Usually, historical data is used to train a model of either the standard behaviour of the machine, or – if enough example cases have been recorded – a model of the deviant behaviour right before the failure. These models are then used on live data to determine whether the machine is operating within standard parameters, or, in the second case, if the operating characteristics are similar to the failure scenario. If the model is trained correctly, it will give an alarm in due time.

In our example case, the machines are cargo trains. These trains are pulling up to 3000 tons of cargo, so a lot of parts are prone to deterioration effects. *DB Schenker Rail* started the *TechLok* project in 2011 with the goal to discover underlying processes of specific failures to implement counter-measures. In [2] we evaluated the process of building such a model based on a specific failure type. In this paper we are focussing on the *power converter*, a unit converting high-voltage electricity from the power lines to be used to drive the electric train motors.

This paper is organized as follows. Section 2 will give an overview of possible scenarios that are suited for predictive maintenance. In Section 3 we elaborate on the methods used in data driven scenarios, before introducing our own prototype for *power converter* failure prediction in Section 4 and 5. Finally show the results and give an outlook on improvements and future research in Section 6.

2 Scenarios

A prerequisite for the implementation of a successful PM system is the availability of a model. There are different approach to construct such a model: Either we learn it from data we collected, or we build it based on our understanding of the physical basics of the system. Another option is to rely on experience with the system, which we can turn into an expert model or a fuzzy model. Physical models are usually based on complex mathematical calculations that try to perfectly simulate the real system. While this is clearly a desirable situation, it is difficult to construct such a model in the first place and may take a long time. Expert models rely on the knowledge of people that have worked with a machine and know all of its behavioural patterns, such that they can extract rules, recommendations and guidelines based on their knowledge.

In order to be able to construct a data driven model, data that resembles the machine status in some way is required. This can be a stream of measurements from sensors embedded in the machine, or more abstract information, for example log-files, that allow conclusions on the underlying state of the machine. Log-files in this case are sort of an expert system, because they were programmed by people that implemented rules such as *When X occurs, then issue message Y* based on a certain reasoning. Still, data driven methods can be able to extract knowledge from these expert systems that humans themselves can not comprehend or derive, because it is too complicated or there is just too much data. The downside of these models on the other hand is that they may work fine, but we might be unable to comprehend why. This is especially true for models

constructed with advanced techniques like deep neural networks, which are very hard to interpret.

In conclusion, the different model types are not mutually exclusive. On the contrary, it might help to have all of them available to get better predictions from their combined abilities.

General information regarding predictive maintenance can be found throughout various research fields and in practitioners books (c.f. [10],[7]) as well as books about the financial and management aspects (c.f. [5]). An overview of various PM methods is given in [8] in more detail.

In Fig. 1 we show a flowchart with which we can assess what type of machine learning is suitable for a given predictive maintenance scenario. In this paper we will elaborate on supervised and unsupervised learning and further discuss the possibilities and pitfalls that occur in data-driven scenarios.

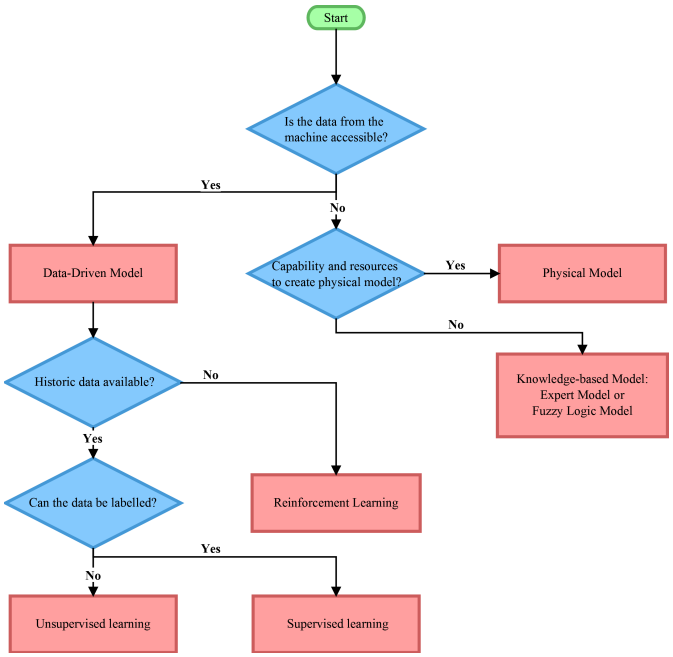

Fig. 1. Assessing a PM scenario

3 Data Driven Methods

In this section we will elaborate on the different types of data that usually exist when dealing with complicated machinery. After that we take a look at various types of challenges that are of interest for us and show some pitfalls that might occur. In recent

years data around big machinery is constantly growing, along with the complexity of the machines themselves. Deriving knowledge-based or physical models has become increasingly complicated, hence data centred approaches have become more popular.

3.1 Types of Data

As mentioned before, data usually comes in one of two forms: Either as continuous series of one or multiple sensor-readings (e.g. temperature, vibration), or as a log-file of system messages.

Time Series of Sensor Readings Monitoring a machine all of the time to its fullest extent, maybe even get readings on all values multiple times per second, is the dream scenario from a data point of view. With this amount of data we can extract or filter out exactly the measurements needed to build a model for a specific scenario.

Despite from this being practically utopical, it brings up a lot of challenges. How does the data get stored? Where does it get stored? In the machine itself, or in a central database? Is there even a connection to that central database? Do we have access to the data so we can do real-time evaluations? How big will this database get over time? Is this expensive?

Most of the times the answer to the last question will be yes. Either caused by a lot of measured variables, by the number of machines that have to be monitored, or by the combination of both. Large databases are needed to cope with these amounts of data properly in terms of storage and retrieval. Measuring all the possible data also causes further problems when building a model, such that it is probably unclear which are the relevant measurements for the selected situation. This can be solved with automated *feature selection* algorithms, but their usage itself requires thorough understanding and is not trivial.

Event Based Data In some situations, there is no availability of sensor data. This may be because it was not wanted by the manufacturer of the machine, or because the machines themselves were built in times where storage and computing power was restricted. Nevertheless, when dealing with (modern) machinery that is somehow automated or operated via a computer, sensors will be integrated in the machine for controlling purposes. Most of the time the machine will provide a log-file. In a log-file the machine records activities, warnings and failure messages. We will refer to these recorded messages as diagnostic messages (DM).
Diagnostic messages are a caused by internal rules being triggered, which might result from internal evaluation of actual sensor readings. These sensor readings are usually provided in addition to the error message, so that an engineer can identify the underlying condition causing the error. The disadvantage of this type of data is, that through its event based nature, there is only information recorded when something actually happens. When we want to gather lots of information about e.g. a specific temperature reading this might be a problem, because there is no way of retrieving continuous readings over an extended timespan.

A possible advantage of this type of data might be, that just the analysis of these messages w.r.t. amount and frequency can lead to a conclusion, e.g. significantly higher amount of certain messages showing abnormal operation and hint on potential failures (c.f. [2]).

3.2 Types of Learning

The field of machine learning is grouped into various sub-categories, based on what type of data and additional information exists, and what one tries to accomplish with it. In this section we will explain a few machine learning problems that are relevant to our research as well as the approaches used to solve them.

Classification and Supervised Learning Classification is the task of predicting a class for a given data instance, the so called *label*. In order to do this, a so called classifier is used. A set of rules for example can act as a classifier. This ruleset decides based on the attributes of the instance, which class it most likely belongs to, e.g. *If Height* $>$ *2 AND width* $<$ *1 THEN class = Tree*.

Usually, classifiers are trained in a *supervised* fashion. This means, a learning algorithm is trained on a labelled dataset (the labels are provided manually) to extract the essential differences between the given classes and decide which class an instance belongs to. Common classes of classification algorithms are: *Rule learners*, *Decision tree learners*, *Bayes Classifier learners*, *Support-Vector-Machines* and *(Deep) Neural Networks*.

Clustering and Unsupervised Learning When dealing with datasets that are not labelled, clustering algorithms are used to establish similarities between instances and group them into clusters. This can be used to gain more insight in the data. In some cases, these calculated clusters can be mapped to real-world classes and thus let us gain knowledge. Depending on the complexity of the algorithm and the data, clustering can be computationally heavy and the results hard to interpret.

Popular algorithms for clustering are *K-Means* (Distance-based), *Expectation Maximisation Clustering* (Probability-based) and *DBSCAN* (Density-based).

Anomaly Detection Anomaly detection (also known as outlier detection) tries to identify instances in a dataset that do not fit the existing patterns. This can be caused by unusual or false measurements and other deviations, as well as novelties that were never seen before. Outlier detection can be solved based on cluster analysis as well as distance or density based approaches with adapted clustering algorithms.

3.3 Problem Types

When performing maintenance, e.g. in regular intervals, the goal is always to assure the flawless functionality of a machine by checking various parts. Depending on the importance of these parts, they will be maintained more or less frequently. For parts

that are prone to physical deterioration, this frequency is dependent on the design of the part, e.g. it is engineered to be replaced every two years, but can also be influenced by experiences that have been made maintaining the part in the past.

In order to perform predictive maintenance, a deterioration process has to be present. Damages that are caused abruptly, e.g. through external influences, can not be predicted. Besides the necessity of a deterioration, it is also important that this deterioration, or side effects thereof, are (i) picked up by sensors and (ii) recorded for further analysis. Deterioration speed affects the *prediction horizon*, meaning the time difference between a possible prediction and the actual failure. Slowly deteriorating systems can be identified earlier, allowing for better scheduling of replacement or reparation arrangements, whereas spontaneous deterioration is limited in this regard. Although for slowly deteriorating systems the prediction horizon is usually larger, it does not necessarily mean the system decays in a linear fashion. The behaviour can also resemble a series of rather abrupt deterioration steps, or change depending on the circumstances (e.g. outside temperature or weather conditions).

Beside the underlying physical processes that may vary for each type of deterioration and hence show in different ways, from a data point of view there are also differences and challenges to be overcome, which we will adress in the following section.

3.4 The Example Inconvenience

In general, companies try to avoid failure of their equipment as much as possible. This generates a problem when trying to learn a model to predict failure. Usually there are not many cases available where the complete history of the failure is recorded up until the moment of breakdown. For supervised learning this is problematic in such a way that it complicates the model training process due to lack of examples. Based on a severely skewed example distribution with only a few *positive* failure examples, the model training has to be adapted. Techniques such as Over- and Subsampling, class-weight balancing as well as other methods can be used to help solve these problems, but should always be used with caution. In the further sections we will use subsampling based on various ratios to reduce the distribution gap to a fixed factor. Which factor yields the optimal results will be discovered experimentally.

4 Building a Classification Model

In this section we propose a two-step approach for classifying tours of cargo locomotives in an *online* manner. A tour resembles the timespan between starting the locomotives' electric systems and shutting it down. In our case we only look at tours that are at least two hours long with over 100 kilometres driven. We will train a classifier based on the diagnostic data instances, and then introduce a meta classifier to combine individual predictions to an assessment of the entire tour. First we describe the labelling process we used for our supervised learning approach. After that, we elaborate on intermediate results that we obtained by applying the learned models on the data and build the meta classifier for tour-level classification based on the observations of the results.

4.1 Converting Diagnostic Data to Instances

Diagnostic data is collected on the locomotive in the form of a logfile, in which all events occurring in the various systems of the locomotive are recorded. In total, there are 6909 different event types, the so called *diagnostic messages* (DM). Diagnostic messages can contain status information, warnings or specific error messages. They have two timestamps, one depicting the time when the message started (*from*), and one when it ended (*to*). This is a specific feature to diagnostic messages that other event-log systems might not have.

In our pursuit to classify tours in an online scenario with *live* data, we decide to keep the overhead for data transformation into our desired form as small as possible. Therefore we look at each diagnostic message as a boolean value, which is either *active* or *inactive* for any specific point in time. In the historical data this can be decided based on the *from* and *to* values that are set for each DM.

Table 1. Example of status vectors as binary instances with relevant changes

Instance \ Code	C_1	C_2	C_3	C_4	...	C_{n-2}	C_{n-1}	C_n
Instance 1	1	0	0	1	...	0	1	0
Instance 2	1	1	0	1	...	0	1	0
Instance 3	0	1	0	1	...	0	1	0

Preparing the data this way, we receive a set of instances per tour. The size of this set depends on the duration and the circumstances. An example is shown in Tab. 1.

4.2 Labelling with variable Window Size

Since the data is unlabelled we only have information about when the incident happened. To train the classifier, we need to artificially generate labelled instances to learn upon. This is a crucial aspect with influence on the quality of the resulting classifier. We use a modified version of the labelling method suggested by Létourneau et al. [4]. This method uses a windowed labelling approach: We label the instances such that they are labelled *positive* in a window before the failure occurs, and *negative* otherwise. Because we do not know which the indicative instances are or how they are different from non-indicative instances, we define an integer w as the duration of this window, the so called *warning epoch* (see Fig. 2). We will determine the optimal value of w experimentally.

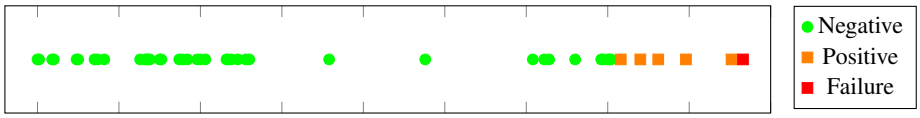

Fig. 2. The labels assigned to instances of an exemplary *incident* tour

We are looking at a failure that has a short prediction horizon, meaning that the selection of the *positive* instances is crucial to the success of the method. Otherwise our classifier will not be able to build an adequate model. The number of *positive* instances depends on the size of the labelling window w. We analysed the quantity of labelled instances in the *incident* tours by using window sizes from 60 to 9600 seconds (more precisely: 60, 150, 300, 600, 1200, 2400, 4800, 9600) in order to determine plausible values for w. For the lower end of the window size spectrum, e.g. 60 and 150 seconds, we receive low numbers of positive instances, so that some of the tours contribute no instances to train the classifier upon.

We are also dealing with a heavily skewed dataset in terms of *positive* and *negative* instance numbers since we have very few incidents but a large number of normal tours. With this few instances to train upon, we expect poor classification performance from the models trained with these parametrisations in the evaluation.

By training the classifier this way, we do not expect to achieve a perfect classifier, but rather create an *anomaly detector* that can be deployed with little effort on the given systems. More complex anomaly detection methods are available in the form of e.g. One-class SVM [6], but they can not be implemented in this proof of concept prototype.

4.3 Instance level Classification

After we constructed the models by the procedure given in Section 4.2, we apply them on the given tour data. Since the model is trained to recognize the instances right before the failure, and we assume those instances show a different behaviour than while operating normally, it will classify instances as *positive* when they match the pattern of the almost-failure instances. In an ideal scenario, the model would classify tour instances as *negative*, and switch to *positive* when nearing the point of failure, much like the data it has been trained with.

Unfortunately, this behaviour occurred rarely when we applied the model. Instead, the model would give predictions like those shown in Fig. 3. We found five principal types of behaviour when inspecting the classification results of a sample of tours.

- **T1** - The classifier gives constant positive results when nearing the point of failure.
- **T2** - Spontaneous spikes give hints on faulty behaviour, but there is no consistency.
- **T3** - A cluster of spikes, or multiple clusters, with variances in density/length.
- **T4** - A positive phase is followed by a negative phase.
- **T5** - Purely negative, no indication is given.

If we want to make a prediction for a complete tour based on the results of the instance level classifier, we can do this in a naive way by assuming that if a single instance is classified as *positive*, the tour will be classified as *incident*. If the classified tour is an actual *incident*, the types **T1-T4** would give true positive results, and a false negative in case of **T5**.

Besides the classification of *incident* tours, we also classified *non-incident* tours using the same model. Unfortunately, the behaviours shown in Fig. 3 could also be observed there, although **T1-T4** occurred less often. The naive method would create false positives in cases **T1-T4** when making predictions for the whole tour. To solve this on

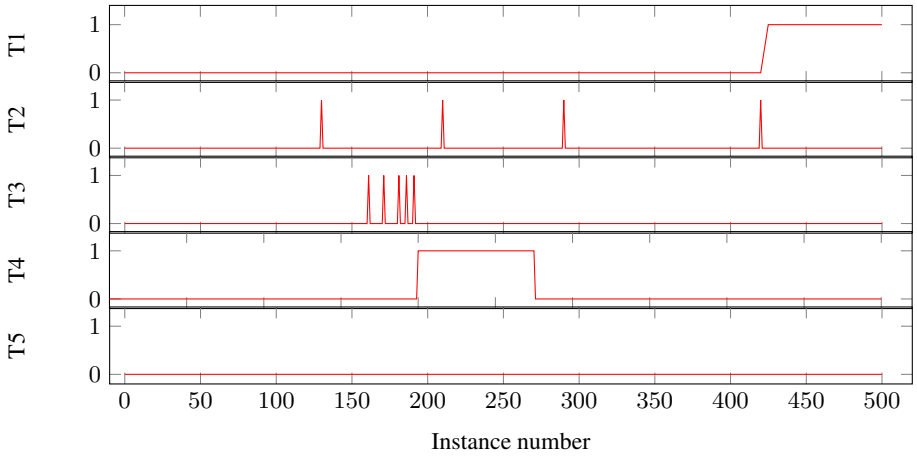

Fig. 3. Instance level classification behaviour in tours: principal types (Incident at 500)

the instance level, we need more precisely trained models. This issue was anticipated: with only 40 examples of PCF, a well-performing generalised model was not to be expected.
Depending on the parametrisation of the model, types **T1** and **T4** did not occur at all in our sample set of *non-incident* tours. Behaviour type **T2** on the other hand occurred frequently. We will propose an advanced meta classifier to prevent these occurrences from creating *false positives* and improve tour-level classification in the next section.

4.4 Tour-level Meta Classification

In this section we will propose two methods for the meta-layer of the prediction process. The baseline tour classifier *SimpleDirect*, and a method which will improve tour-classification performance beyond the limitations of *SimpleDirect*.

The *SimpleDirect* Tour Classifier. We use the *SimpleDirect* (SD) method as a baseline tour classification method. The classifier SD is defined such that it uses an instance classifier CI. If CI classifies an instance as *positive*, SD will classify the tour as *incident*.

The instance classifier gives us an indication based on a single instance as class: either *positive* (as in: failure is likely) or *negative*. It is un-intuitive to base the classification of a whole tour on a single instance. We would rather look at a certain number of instances and make the decision on multiple results of instance classifications. We present a naive approach to this problem which improves on the *SimpleDirect* schema.

Meta Classification by Percentage Threshold. This method requires a threshold $t = positive/total$ to be surpassed to predict the *incident* label for the whole tour. Instance level classification is applied separately to the instances in the order of their arrival

time, and the result is used to calculate the threshold. The method requires a minimum number of seen instances w.r.t. the threshold ratio. This has the positive effect that it reduces false positives in the case of **T2**-type instance classifications, but it also limits the classifier to make a very early decision caused by the necessity of a minimum of seen instances. We will take this into account when evaluation the performance in Section 6.

5 Prototype for Power Converter Failure prediction

In the following, we will introduce the classifiers, elaborate on the methods we used to help improve the problem with imbalanced classes, explain our chosen evaluation method and state all variants of the tour-level meta classifier we used. Finally, we will explain the evaluation method.

5.1 Parametrisation of the Classifiers

Our goal is to create a lightweight prototype installation of the prediction system, which means we have to rely on classifiers that can be implemented on the given systems. We decided to go with two basic classifiers, JRIP (as the WEKA [13] implementation of the RIPPER rule learning algorithm [1]) and J48 (WEKA implementation of C4.5 decision tree learner [9]). A ruleset or a decision tree is easy to implement on a system that only allows a low-level scripting language, and therefore the right choice for our endeavours. Furthermore, they are interpretable by humans. This is especially useful for the engineers to draw conclusions from the learned rules or trees and helps justify or discard a model.

5.2 Sampling and Balancing

As a result of the chosen labelling process we have a set of instances with a highly skewed class imbalance of up to 99 to 1. Training a classifier on this kind of data might give suboptimal results, because classifiers will focus on the majority class (and subsequently be correct more than 99 % of the time). We will apply the Subsampling technique *SpreadSubsample*[1] in order to improve classifier performance:

Subsampling rebuilds the training set, such that a certain ratio between the rarest and the most frequent class is realised. For example we use this to create a 15:1 spread ratio of *negative* to *positive* class. Of course by this measure the number of *negative* instances in the training set will be reduced, which might have an influence on learning the class in some classifiers.

We used both of the classifiers with their default WEKA configuration. The default configurations provide solid classification performance, which we can later optimise, but at the moment we will concentrate on the other parameters.

[1] http://weka.sourceforge.net/doc.stable/weka/filters/supervised/instance/SpreadSubsample.html

5.3 Leave One Tour Out Evaluation

Our data consists of all 40 *incident* tours, and we add a random sample of 300 *non-incident* tours from our pool of tours that have a duration of more than 2 hours and 100 km.

For evaluation we use a leave-one-out cross-validation method, i.e. we train the model on 339 tours and apply it to the remaining one, for each of the 340 tours. This method is a version of the *leave one batch out* method described in [3], which leaves a batch of instances out of the training data. In our case, a batch consists of all instances from the left-out tour. We want the maximum possible number of *incident* tours in the training set, so that the classifier has the best chance to learn a proper model. By this we assure that at least 39 *incident* tours are in each training set, which gives the best possible conditions for the classifier.

At the instance level, the 40 *incident* tours contribute 10903 and the 300 *non-incident* tours 58919 instances to the train/test data. The number of positive instances depends on the w value.

5.4 Variants

We used the following parameters and values in combination to find the optimal result:

The labelled Data. We used the process described in Section 4.2 in two variants. The first variant excludes the instance that contained the actual failure, the second includes it.

Labelling Windows. We used 7 labelling window sizes as described in Section 4.2 from *300 s* to *9600 s* in order to empirically find the optimal window.

Class Balancing. For balancing we used Subsampling (see Section 5.2) in 3 variants with spread-factors of 15, 30 and 50 and a variant with no balancing.

Tour-level. In order to classify complete tours we applied the *percentage threshold* (see Sec. 4.4) in 6 variants (*2, 5, 10, 20* and *30 %* threshold) as well as *SimpleDirect* (*1 hit in total*). Since the result is determined by this parameter in combination with the instance level classifier we used a wide spread of values.

As stated before, we used both J48 and JRip as classifiers, resulting in a total of 1260 combinations of parameters. For each of these combinations we applied the leave-one-tour out evaluation method, resulting in around 10000 hours computing time for the complete evaluation. Unfortunately, caused by time constraints, we were not able to compute multiple runs with different sample sets of *non-incident* tours.

6 Experiment Results

In this section, we will describe the baseline of our experiment, show the results we achieved, and compare the enhanced meta classifier to the *SimpleDirect* baseline.

6.1 The Accuracy Baseline

Since we are targeting the classification of tours and have no related method to use as a baseline, the ratio of the tour classes will be the starting point. We are using a dataset with 40 *incident* tours and 300 *non-incident* tours, so a classifier that always chooses the majority class would achieve an accuracy of $88.2\ \%$, albeit with no *true positives*.

6.2 Relevant Results

Due to the amount of parameter combinations we will not be able to show all results here, instead we show a pre-filtered set based on the criterion, that the *minimum time-to-failure* (mTTF) should never be zero, which would mean that a tour was classified based on the last instance in the tour, and be of no practical use. Furthermore, we sort the remaining results by accuracy. We want as few false positives as possible, but at the same time a maximum of true positives. After the filtering we are left with 574 resulting classification variants.

Caveat. In our evaluation we assume that *false positives* are costly. Unfortunately we do not know the real cost of PCF and false alarms, so we can not base our findings on it. With over 90000 tours in our dataset, even a 1 % *false positive* rate would lead to 900 locomotives having to be checked. Therefore we want to find a method that creates little to no false alarms. We also asked the people that will have to use this system, and they would rather work with a method that has a higher confidence and less hits than a classifier that creates many false alarms.

The results (Tab. 2) are ranked by the criteria mentioned above. The columns contain the following information:

1. *Rk*: Rank as determined by our sorting criteria
2. *f.I.*: Dataset contains the *failure instance* (*f.I.* = 1) or not (*f.I.*=0)
3. *Window*: Size of the specific labelling window
4. *Classifier*: The classifier used
5. *Balancing*: Balancing method (Subsampling or None)
6. *Meta Classifier*: Parametrisation of the meta classifier, or *SimpleDirect* (SD)
7. *Accuracy*: Percent correctly classified instances
8. *aTTF*: Average TTF in correctly classified *incident* tours in hours
9. *mTTF*: Minimum TTF in correclty classified *incident* tours in minutes
10. *TP, TN, FP, FN*: True positive, true negative, false positive and false negative tours

Table 2 shows that we can achieve multiple combinations with no false positives. On the top ranks we have a maximum of 12.5 % (5 of 40) true positives. Although this does not seem impressive at first sight, with an expensive failure like the PCF this can be a valuable asset used for cost reduction. All the methods in the top ranks show an *average time-to-failure* high enough for a person to react. But for example at Rk. 9 the reaction time could be too short to react properly because the *mTTF* is only one minute.

As far as the optimal labelling window is concerned, the best performing configurations were built on the 2400 s window size. Although we can not precisely determine

Table 2. Top results with $mTTF > 0$ ordered by Accuracy

Rk	f.I.	Window	Classifier	Balanc.	Meta	Accur.	aTTF	mTTF	**TP**	TN	**FP**	FN
1	0	2400	JRip	subs50	2perc	89.57	8.91 h	141.22 m	5	304	0	36
2	0	2400	JRip	subs50	5perc	89.57	8.54 h	138.47 m	5	304	0	36
3	0	4800	JRip	subs30	20perc	89.57	11.55 h	99.02 m	5	304	1	35
4	0	4800	JRip	subs30	30perc	89.28	14.89 h	62.62 m	3	305	0	37
5	0	4800	JRip	subs50	30perc	89.28	12.33 h	62.62 m	4	304	1	36
6	1	2400	J48		20perc	89.24	10.89 h	99.43 m	3	304	0	37
7	1	2400	J48		30perc	89.24	9.64 h	86.03 m	3	304	0	37
8	1	2400	JRip		2perc	89.24	8.50 h	141.23 m	3	304	0	37
9	1	2400	JRip	subs15	20perc	89.24	15.46 h	1.03 m	8	299	5	32
10	1	2400	JRip	subs50	30perc	89.24	6.29 h	86.03 m	3	304	0	37
11	0	2400	JRip		2perc	88.99	7.94 h	141.22 m	3	304	0	38
12	0	2400	JRip		5perc	88.99	7.75 h	138.47 m	3	304	0	38
13	0	2400	JRip		10perc	88.99	7.35 h	135.70 m	3	304	0	38
14	0	2400	JRip		20perc	88.99	5.79 h	1.02 m	3	304	0	38
15	0	2400	JRip	subs50	10perc	88.99	7.10 h	135.70 m	3	304	0	38
16	0	2400	JRip	subs50	20perc	88.99	5.35 h	1.02 m	3	304	0	38
17	0	4800	JRip	subs15	30perc	88.99	14.85 h	66.90 m	4	303	2	36
18	0	60	JRip	subs50	10perc	88.95	18.87 h	48.78 m	4	302	1	37
19	1	2400	JRip		5perc	88.95	10.00 h	138.48 m	2	304	0	38
20	1	2400	JRip		10perc	88.95	9.56 h	135.72 m	2	304	0	38

...

Table 3. Top 5 results with $mTTF > 0$ and *SimpleDirect* meta classifier

Rk	f.I.	Window	Classifier	Balanc.	Meta	Accur.	avgTTF	minTTF	TP	TN	FP	FN
459	0	2400	J48		SD	12.17	11.96 h	53.70 m	42	0	303	0
460	0	2400	J48	subs50	SD	12.17	11.96 h	53.70 m	42	0	303	0
461	1	1200	J48		SD	12.17	11.42 h	45.02 m	42	0	303	0
462	1	1200	JRip		SD	12.17	11.42 h	45.02 m	42	0	303	0
463	1	1200	J48	subs15	SD	12.17	11.42 h	45.02 m	42	0	303	0
464	1	1200	J48	subs30	SD	12.17	11.42 h	45.02 m	42	0	303	0
465	1	1200	J48	subs50	SD	12.17	11.42 h	45.02 m	42	0	303	0
466	1	1200	JRip	subs15	SD	12.17	11.42 h	45.02 m	42	0	303	0

why this works best, we assume it's a combination of the number of instances and the relevance of the instances they cover. Most of the top 20 results also include some version of the SpreadSubsample filter, but the variants without balancing are not far behind (e.g. Rk.6). All of the top results perform above baseline in terms of accuracy, although not by much.

We can also see that the *tour-level* meta classification increases the classification performance compared to *SimpleDirect* (Tab. 3). The first configuration that uses *SimpleDirect* is ranked on position 459 in the results. It performs substantially less good in terms of accuracy because it can not differentiate the tour classes at all. This is also the case for all other SD variants.

The percentage threshold method has the disadvantage that it has to gather a certain number of instances before it can make a decision. But as we see in Tab. 2 and 3, this is a necessity for a useful classification and the TTF values are still usable in real world applications.

6.3 Conclusion and Outlook

We have created the basis for an easy-to-implement prediction prototype that can be realised even on systems with limited capabilities. In our case, a ruleset with a meta classifier built on top shows good recognition performance, while creating no *false positives* on our dataset. This is especially useful to create trust in the method among the engineers and people that are supposed to use it on daily basis.

The next step will be the realisation of the prototype as well as research methods with higher complexity that improve upon the achieved results. At the moment we apply fixed labelling windows to all tours. This might not yield the optimal result for each of the tours. It is likely that we have to decide upon other criteria to improve the labelling, in order to train a better instance-level classifier. Also, the tour-classification is naive at the moment. We will try to learn another classification model on the output of the instance-level classifier w.r.t. the tour class, so that the results can be improved and we achieve more hits and less false alarms. Further improvements could be achieved by combining multiple classifiers and their decisions on the instance as well as the tour level. Finally, we will also apply more complex learning algorithms on the data. While the practical use may be limited at the moment, it is useful to see if the results can be improved.

Further approaches would include treating this problem as a multi-instance problem [11], especially regarding the question, if the *online* character of this situation can be handled in a multi-instance setting. Another way of approaching the topic would be via pattern learning [12], e.g. by discovering sequences of diagnostic messages that lead to failure, and of course apply *Deep Learning* methods (e.g. Recurrent Neural Networks for time series classification.

References

1. William W Cohen. Fast effective rule induction. In *Proceedings of the twelfth international conference on machine learning*, pages 115–123, 1995.
2. Sebastian Kauschke, Immanuel Schweizer, and Frederik Janssen. On the challenges of real world data in predictive maintenance scenarios: A railway application. In Sebastian Görg, Gilbert Müller, and Ralph Bergmann, editors, *Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB*, pages 121–132. CEUR Workshop Proceedings, October 2015.
3. Miroslav Kubat, Robert C Holte, and Stan Matwin. Machine learning for the detection of oil spills in satellite radar images. *Machine learning*, 30(2-3):195–215, 1998.
4. Sylvain Létourneau, Fazel Famili, and Stan Matwin. Data mining for prediction of aircraft component replacement. In *IEEE Intelligent Systems Jr – Special Issue on Data Mining*, pages 59–66, 1999.
5. Joel Levitt. *Complete Guide to Predictive and Preventive Maintenance*. Industrial Press, 2011.
6. David Martinez-Rego, Oscar Fontenla-Romero, and Amparo Alonso-Betanzos. Power wind mill fault detection via one-class v-svm vibration signal analysis. In *Proceedings of International Joint Conference on Neural Networks*, 2011.
7. R Keith Mobley. *An introduction to predictive maintenance*. Butterworth-Heinemann, 2002.
8. Ying Peng, Ming Dong, and MingJian Zuo. Current status of machine prognostics in condition-based maintenance: a review. *The International Journal of Advanced Manufacturing Technology*, 50(1-4):297–313, 2010.
9. J Ross Quinlan. *C4. 5: programs for machine learning*. Morgan Kaufman Publishers, Inc., 1993.
10. Cornelius Scheffer and Paresh Girdhar. *Practical machinery vibration analysis and predictive maintenance*. Elsevier, 2004.
11. Ruben Sipos, Dmitriy Fradkin, Fabian Moerchen, and Zhuang Wang. Log-based predictive maintenance. In *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1867–1876. ACM, 2014.
12. Risto Vaarandi et al. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 2003 IEEE Workshop on IP Operations and Management (IPOM)*, pages 119–126, 2003.
13. Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.