Advances in Predictive Maintenance for a Railway Scenario - Project Techlok

Technical Report TUD–CS–2015–1256
Version 1.0, October 31st 2015

Sebastian Kauschke
Knowledge Engineering Group/Telecooperation Group, Technische Universität Darmstadt

Frederik Janssen
Knowledge Engineering Group/Telecooperation Group, Technische Universität Darmstadt

Immanuel Schweizer
Telecooperation Group, Technische Universität Darmstadt

Technical Report TUD–KE–2015–01
FG Knowledge Engineering, TU-Darmstadt

Telecooperation Report No. TR–16
The Technical Reports Series of the TK Research Division, TU-Darmstadt
ISSN 1864-0516
http://www.tk.informatik.tu-darmstadt.de/de/publications/

Advances in Predictive Maintenance for a Railway Scenario - Project Techlok

Sebastian Kauschke KAUSCHKE@KE.TU-DARMSTADT.DE
KNOWLEDGE ENGINEERING GROUP & TELECOOPERATION GROUP
TECHNISCHE UNIVERSITÄT DARMSTADT, GERMANY

Frederik Janssen JANSSEN@KE.TU-DARMSTADT.DE
KNOWLEDGE ENGINEERING GROUP & TELECOOPERATION GROUP
TECHNISCHE UNIVERSITÄT DARMSTADT, GERMANY

Immanuel Schweizer SCHWEIZER@CS.TU-DARMSTADT.DE
TELECOOPERATION GROUP
TECHNISCHE UNIVERSITÄT DARMSTADT, GERMANY

Contents

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. A successful predictive maintenance process for a machine can help with preventing failures, 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. An overview of various PM methods is given in Peng et al. (2010).

The DB Schenker Rail *techlok* project is centered around the idea of system failure prediction on trains based on their diagnostic data. The first steps towards a predictive model were made in 2014 based on the diploma thesis of Sebastian Kauschke (Kauschke et al. (2014)). We leveraged the event-based structure of the diagnostic codes to find patterns in the frequency of the events which indicate a failure scenario. With the help of a supervised machine learning approach we built a classification model that analyses the diagnostic data on a daily basis.

In this document we will show the advances of our research during the last twelve months. We worked with more data sources — information regarding the workshop activities (SAP ISI Database) and failure reports (DES database) — and a longer period of data recordings (two years). With this new information we were able to more precisely determine when a specific kind of failure occurred, and therefore enhance the ground truth for the historic data which we train the classification model upon. Furthermore, we integrated the issue of *Fehlzuführungen* (Unnecessary layovers).

One of the main tasks was to differentiate unnecessary layovers compared to necessary ones. For this task, recognizing them is vital for the training of the model. For the general prediction of impeding failures, it is also advantageous to know unnecessary layovers, because they will affect the ground truth.

In order to achieve these tasks, we generate a descriptive snapshot of a train's diagnostic data for each day, a so called *instance*. These instances are multidimensional vectors and consist of so called *features*. Each feature captures a certain aspect of one of the variables we observed on this train during that day. There may be several hundred up to thousands of features in an instance. Given an instance for each train and day created by the data records we obtained, we can use machine learning methods to learn a model. This model will then be able to classify new instances.

In the experiments we executed during the process, we have not yet come up with results that can help us resolve the abovementioned tasks. This may sound disappointing compared to the results of 2014, but is caused by the increase of available information. We are looking into obtaining even more data sources to further enhance our knowledgebase, as well as research different methods of building comprehensive and functional models.

In the remainder of this document we will give a conclusive summary of the steps we have passed so far. Starting with the description of the general task and the data-sources we were supplied with, we elaborate on the basic problems that were researched, continue with the technical explanation of the methods and conclude with experiments and results thereof. Finally, we give an outlook on planned future work.

2. Data

In this section, we will give a short introduction to the three information sources that were available, and make assumptions about the quality and the completeness of the data. In comparison to our effort in 2014 (Kauschke et al. (2014)), were we relied solely on the diagnostic data, we were supplied with more data from the DES and SAP ISI databases (see Fig. 1) in a wider period of two years (2013 and 2014).

This enabled us to gather more precise information and tackle a variety of new issues.

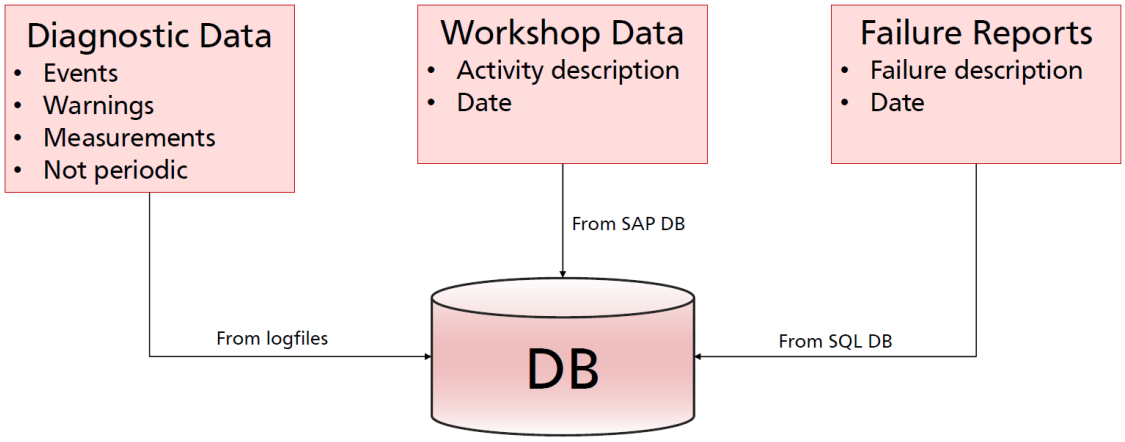

Figure 1: Data sources

Given the combinations of these information sources, we will then filter the occurrences of two specific failure scenarios and determine the exact date of the failures (cf. Sect. 4).

2.1 Diagnostics Data

The diagnostics data is recorded directly on the train in the form of a logfile. It records all events that happened on the train, from the manual triggering of a switch by the train driver to warning and error messages issued by the trains many systems.

We have access to data from the BR185 model range of trains, the *BR185*. This range was built in the 1990s. It has internal processing and logging installed, but the storage capacity is rather limited. Back then, predictive maintenance was not anticipated, and the logging processes were not engineered for this application. This becomes abundantly clear, when we consider the steps necessary to retrieve the logfiles from the train. It has to be done manually by a mechanic, each time the train is in the workshop for scheduled or irregular maintenance tasks.

In the two years we are using as historic data, around 21 Million data instances have been recorded on 400 trains.

2.1.1 Diagnostic Codes and System Variables.

As already mentioned, the diagnostic data is stored in the form of a logfile. It consists of separate diagnostic messages, each having a code to identify its type. When we refer to a diagnostic code, we usually mean a message with a specific code. In total there are around 6.900 possible diagnostic codes. Each time a specific system event occurs, a diagnostic message with the respective code is recorded.

A set of system variables is attached to each diagnostic code. Those are encoded as Strings (Fig. 2). The variables are not monitored periodically, but only recorded when a diagnostic message occurs, since the whole system is event based. Which variables are encoded depends on the diagnostic message that was recorded. This implies that some variables will be recorded rarely and sometimes not for hours or days. Overall, there are 2291 system variables available: simple booleans, numeric values like temperature or pressure and system states of certain components.

The event-based structure complicates handling values as a time series. It requires some sort of binning in order to achieve regularly spaced entities. In our case, a bin size of one day was used. Especially when relying on attributes that involve temperature or pressure measurements it is impossible to create a complete and fine-grained time-series of their values, because they appear too sparsely.

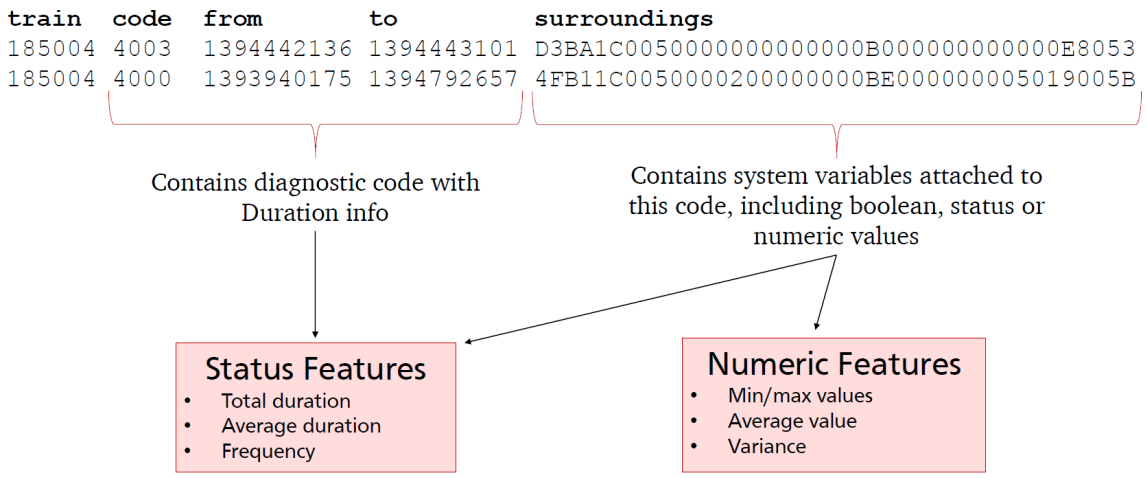

Figure 2: Diagnostic data and attached values

There is another speciality about these diagnostic messages: They have two timestamps (Fig. 2), one for when the code occurred first (*from*), and one for when it went away (*to*). This was originally designed for status reporting messages that last a certain period. Most codes only occur once, so they do not have a timespan, others can last up to days. For example there is code 8703 (*Battery Voltage On*) which is always recorded when the train has been switched on, and lasts until the train is switched off again.

Because of the two entries *from* and *to*, there are usually two measurements of a variable for each diagnostic code entry. To handle these variables correctly, we separate them from the diagnostic messages and use both values as separate measurements. This means, one diagnostic code contains two separate events, one for the *from* timestamp, one for the *to* timestamp, and two sets of system variables, one for each timestamp.

2.2 Failure Report Data - DES Database

The Failure Report Data contains information when a failure has been reported by a train driver. In the driving cabin there is the Multifunktionsanzeige (Multi-function-display: MFD), which shows warnings or error messages. When a critical problem occurs, the information in conjunction with a problem description is shown to the driver. If he is unable to solve the issue, he will call a hotline. The hotline operator will try to help find an immediate solution. In the case of an unsolvable or safety-critical problem, the hotline operator will schedule a maintenance slot for the train and organise the towing if necessary.

The information recorded in the DES database is the date of the hotline call, the stated reason of the caller, and the information if the issue had an impact on the overall train schedule, i.e., the train was delayed more than ten minutes.

The start and end date of the train being in the workshop for repairs will be added to this database afterwards (manually), so there is no guarantee that it is complete and correct. In general, the textual description given by the train driver and hotline operator are free text inputs and not consistent. The easiest possible way of finding out instances of a failure type is to search these text descriptions for certain keywords.

In Fig. 3 a few exemplary entries from the database are shown. There are more fields than shown available in the full dataset.

2.3 Workshop Data - SAP ISI Database

The Workshop Data consists of a complete list of maintenance activities on all the trains. Compared to the Failure Report Data, it is gathered in a more controlled environment. Every maintenance activity is recorded

```
DATE       BAUNR  TEXT            AUSF.BEGIN AUSF.ENDE
1388703600 185176 LZBPZB gestoert 1388617200 1388703600
1388617200 185149 LZB gestoert      1388271600 1388703600
1388617200 185216 LZB defekt      1388444400 1388703600
```

Figure 3: Example entries from DES database (selected attributes)

here, from the replacement of consumables up to the more complex repair activities. Every entry has date information as well as an exact id for each activity predefined in a database. All activities are grouped by a group code, as well as tagged with a price. The information, if a certain action was ”corrective” or a ”scheduled replacement” is also available.

The correct tracking of the maintenance records is necessary for invoices, so it is plausible to assume that these are handled more carefully than the failure report data. However, they are manually entered into the system and it can not be guaranteed that they resemble the exact activities that have been applied to the train.

2.4 Quality issues

All of the three datasets have two issues in common: (1) they may not be complete (missing entries, descriptions or dates) or (2) are filled with false values (wrong dates, typos that make it hard to find a keyword). Whether this is caused by human error, negligence or processes that do not cover enough details is not important, but rather how we deal with these issues.

They may cause problems during the whole process which we may or may not realize at some point. A possible result could be the degradation of results, or — almost worse — lead to very good looking results that then cannot be re-created in practice. In any case, we have to look out for these inconsistencies and question their possible impact in each following step.

2.5 Validity of data

In order to make computations on the given data easier, the concept of *validity* is introduced. If an instance is *valid*, it means that this instance can be used to train a model upon or validate a given model.

Validity is defined as follows: For each train, for each day, an instance is considered *valid*, if

- (i) it contains diagnostic data records
- (ii) the train was not in the workshop
- (iii) the train has driven more than 10 kilometres

If this is not the case, the train might have been in the workshop or was not in use. Especially in the workshops, the train emits plenty of diagnostic codes, probably because of the maintenance crew running diagnostic tools on the train. For the further steps of labelling etc, we only work with *valid* days.

3. Failure Type selection

In order to demonstrate the functionality of our approach we have selected two major failure types, which we chose together with domain experts based on their frequency or how urgently they need to be addressed:

- The *Linienzugbeeinflussung (LZB)* failure
- The *Main air compressor (Hauptluftpresser: HLP)* failure

For both problems the classic *prediction* task — determining when the component will fail — is very interesting in order to reduce cost and follow-up problems caused by the failure.

For the *LZB* problem, the additional and very relevant task of recognizing *unnecessary layovers* (Fehlzuführungen) is of high priority. About one third of reported LZB failures turn out to be false alarms : the train driver overreacted on the displayed error messages. If we can decide based upon the status of the machine, if his concerns are justifiable, this could reduce the amount of unnecessary layovers and therefore cost.

So our target is to predict the failure of two specific types in a complex system containing many components. Therefore we will build a predictive model using a supervised learning approach on historical data, and apply it to new(er) data.

With the given datasets described in Sect. 2, we are going to extract the exact points in time when the failures have happened, so that we can put labels on the instances created from the historical data. We will create features that are descriptive and discriminative for the specific failure, and use them to build one instance per day and train.

We will have to face the following challenges:

1. Deal with large amount of diagnostic data that has unusual properties, i.e., inhomogeneous data distribution[1].
2. Extract a valid ground truth from three given datasets to find out exactly when the failures happened by searching for indications for that type of failure and recognizing unnecessary layovers.
3. Recognize errors, incompleteness and imprecision in the data and derive methods to deal with them.
4. Create meaningful features that emphasize the underlying effects that indicate an upcoming failure.
5. Set up a labelling process for time-series data that enables classifiers to anticipate impending failures early enough to allow for corrective reactions.
6. Define an evaluation procedure that maximises the outcome with respect to given criteria in order to achieve optimal prediction.

For the prediction process as described, it is important to have enough example cases in the historical data. In the remainder of this section we will elaborate on the failure types and tasks that were selected.

3.1 Failure type: Linienzugbeeinflussung

Linienzugbeeinflussung (LZB) — continuous train influencing system — is a centralised system to propagate driving information to the conductor, including speed limits, stopping information and other signals. It is also a security system which issues emergency stops if necessary. Originally, it was invented in the 1960s as an addition to the — back then — state of the art influencing system, to allow for train speeds of up to 200 km/h.

Failure of the LZB system is rather common. With the help of the ISI Database we could detect over 900 repairs in the selected period of two years, making it one of the most repaired systems. Although failure of the LZB is not safety critical, it will slow down the train and hence create follow-up problems.

Due to the LZB being highly connected and problems with it being thoroughly logged as warnings in the diagnostic data, as well as the large amount of occurrences, this problem was chosen for evaluation.

3.2 Failure type: Main Air Compressor

The main air compressor or Hauptluftpresser (HLP) is used to create pressurized air for various applications, including braking. In the given datasets there was a high number of HLP problems recorded in the years 2013 and 2014. The HLP is used only occasionally to fill up an air tank to 10 bar every time the tank's pressure sinks below 8.5 bar. Those events are recorded in the diagnostic data. Furthermore, the actual pressure of the tank is recorded as a numeric value in the status variables. The intention of using this problem case is

1. Most often failures cases of machines are extremely rare. However, extracting instances that describe the regular operation of the machine comes more or less for free. In predictive maintenance, we have to deal with a very skewed class distribution.

that the pressure measurement and frequency of refilling the tank will give substantial information on the deterioration of the compressor.

3.3 Prediction task

The main task to apply to both given problems is the prediction task. The goal of this task is to predict, when the part is heading towards a failure, in order to be able to fix or replace it before the failure occurs. For the prediction task, it is both important to have a reliable model in terms of prediction accuracy — meaning little false alarms and plenty of true positives — as well as to be able to predict enough days in advance to react properly.

3.4 Reduce unnecessary layover task

Especially with the LZB problem, there is another aspect to the situation. A significant amount of reported LZB problems that lead to a workshop layover are then deemed unnecessary. That means, a thorough technical check of the component reveals no malfunction and therefore nothing is repaired. In reality, around 30 % of all LZB related layovers are cancelled, which makes it one of the priorities to reduce the amount of unnecessary layovers.

4. Creating Ground Truth

A necessity in order to address the problems and tasks is to reconstruct the ground truth, to be able to create good labels for a classification process. This means determining when a failure has happened exactly. In this section, we will show how much information is contained in each of the datasets described in Sect. 2 w.r.t. the ground truth, and combine them in such a way that the optimal result based on our current state of knowledge is received.

The information we need to build a model with a supervised learning approach are:

- the day a failure occurred,
- information on when the train was in the workshop afterwards,
- a list of double repairs of the same part within a few days, and
- a list of layovers that were deemed unnecessary.

Without this information being correct, we cannot label the daily instances for each train correctly and the model will be corrupted. The process is as shown in Fig. 4.

4.1 Diagnostic Data only

When we look at the information we can retrieve from the diagnostic data, it seems reasonable to think that we can extract the point in time when the train driver received the precise error message that led him to report the failure.

In reality, however, the messages displayed on the MFD cannot be retrieved from the diagnostic data. They are generated from it with a certain internal programming logic. Unfortunately, the combinations of codes needed to display a certain message is not accessible, therefore it remains unknown how this is done. Given the original documentation, we would be able to identify the causes, which could help find the exact reasons the train driver called the hotline, and also evaluate which of those messages occur before real failures and which before unnecessary layovers (cf. Sect. 4.4).

When we take all diagnostic messages that explicitly state a malfunction of the main compressor into account, a result as depicted in Fig. 5 can be achieved, each square indicating one failure day. Hence, for the train in our example a total of six failure indications are present.

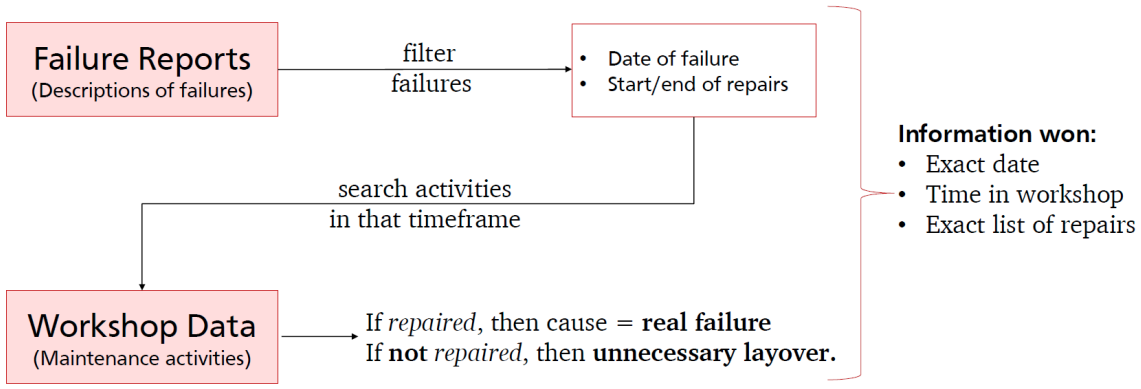

Figure 4: Create Ground Truth

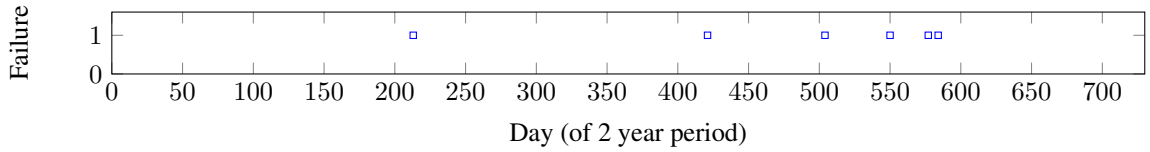

Figure 5: Discovered failures for one exemplary train using only diagnostics data

Because the underlying reasons that caused this messages are unclear, we proceeded by taking further knowledge into account and refine these findings in subsequent steps.

4.2 Workshop data

Using the workshop data, we can determine the point in time when a certain part has been replaced, and if the replacement was corrective or scheduled (mandatory). This greatly improved the identification of the true failures. Still, depending on how maintenance is handled, it only gives us a rough estimate of the point in time the failure actually took place. Note that maintenance procedures are not always carried out directly as the failure happens. Some types of failures require the train to be sent to a certain workshop, as not all workshops are equipped to handle every repair procedure. This may cause some days or even weeks in delay before the train finally is repaired. Therefore, the workshop date is not precise enough for a valid labelling.

A comparison of the extracted failure points can be seen in Fig. 6 depicted as red stars, showing a certain overlap with the findings from Fig. 5, but also completely unrelated entries. On average, red stars are 24 days away from blue boxes. If we only take pairs that are less than 21 days apart into account, the average distance is 6.5 days. But those are only 6 out of 10 instances, which leaves room for improvement and consequently leads us to the next step.

4.3 Failure Report Data

Utilizing failure report data, we are able to increase our understanding of when the actual breakdown has happened. The date of the reporting is noted, and, with high confidence, it also states the correct day. We encountered some irregularities, for example the reporting date being behind the date the train was then brought into the workshop. We can still use this information to narrow down the exact day of the failure, but cannot narrow it down to anything more fine grained than a day, because the precision of the timestamp that is recorded in the reporting system is not high enough. Therefore, we decided to take one day as the

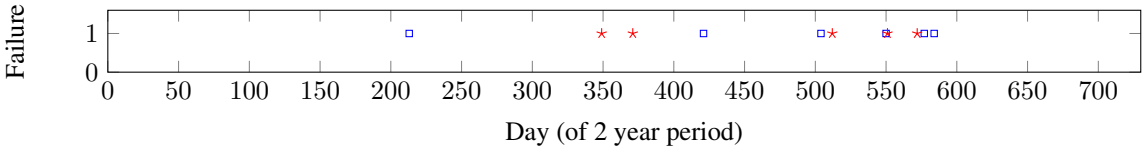

Figure 6: Discovered failures using workshop data (red) compared to Fig. 5 (blue)

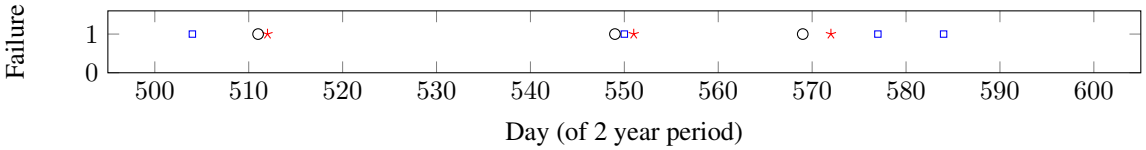

Figure 7: Discovered failures using failure report data (black) compared to Figs. 5 (blue) and 6 (red)

smallest unit a prediction can be done for. Since we expect to predict failures weeks in advance, this is not a problem. However, when failures have to be predicted that evolve within very short periods of time, the proposed method is not suitable any more.

In Fig. 7 we now look at a smaller part of the timeline (compared to Fig. 5 and Fig. 6) from day 500 to 620 (for better visibility). It is obvious that the failure report dates (black circles) are related to the workshop dates (red star), but not always to the diagnostic data (blue squares).

Therefore, we can conclude that only the combination of a failure report and a following repair is truly indicative of a failure. The diagnostic messages seem to indicate failures, but, surprisingly, after most of them the train is not affected negatively. Comparing the failure report dates with the repair dates an average distance of 1.6 days is yielded when events that are more than 21 days apart are discarded.

4.4 Unnecessary workshop layover

Unnecessary workshop layovers mostly happen because of the train drivers concern with safety. As we were told by domain experts, the programming logic that drives the MFDs error and warning messages is usually very sensitive, therefore generating a certain amount of false positives.

This may cause the train driver to trigger unnecessary maintenance actions. In the workshop the mechanics will then check the system, conclude that there is no failure and cancel the scheduled replacement. With the workshop data and the failure report data combined, we are able to differentiate the necessary from the unnecessary activities and exclude them from the pool of failures. This emphasizes the strong need for combining the different data sources by using expert knowledge, as only then high-quality datasets can finally be built. In Fig. 8 the detected unnecessary layovers in comparison to the correct repairs are shown.

4.5 Missed and double repairs

Related to the unnecessary workshop activities are the missed repairs. Sometimes the train might arrive in the workshop with a given failure to check, and the repair crew may not be able to reproduce the symptoms, hence declaring this an unnecessary activity. A few days later the train might get submitted for the same reason again, and often only then the crew will actually repair or replace the component.

This effect has two implications, the first being that the time between those two layovers should not be used to train the model, because it may contain data where it is not certain if the failure is near or not. Second, it is also not clear whether the replacement that was made in the second attempt was actually well reasoned, or the maintenance crew decided to simply replace the part in order to eliminate the interference from happening

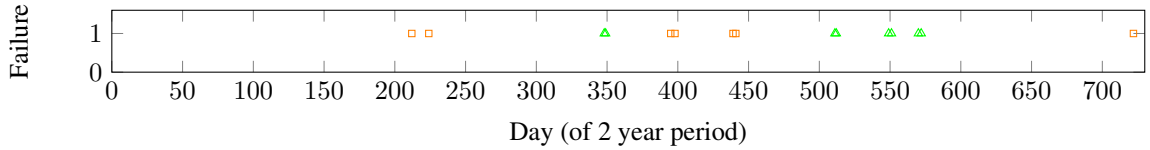

Figure 8: Discovered failures (green) and unnecessary layovers (orange)

Examplary diagnostic code:

```
BAUNR    CODE    FROM        TO
185004   20474   1360001005  1360050266
```

Corresponding system variables (excerpt):

```
VARIABLE                  TYPE  SYSTEM FROM_VALUE TO_VALUE
Laufleistung              LONG  ASG01  1701416    0
Fahrtrichtung Lok         WERT  ASG01  65         2
System                    BDEZ  ASG01  80         0
Traktion                  WERT  ASG01  4          0
Geschwindigkeit           UUUU  ASG01  0          0
Freigabe Stromrichter     BOOL  ASG01  true       false
Zug-/Bremskraftsollw.     BBBB  ASG01  0          0
Bef Stromrichter-Betr     WERT  ASG01  4          0
Zug-/Bremskraftistw.      BBBB  ASG01  0          0
Stromrichter taktet       WERT  ASG01  4          0
Zug-/Bremskr-Reduz (KSR)  UUUU  ASG01  0          0
Fahrmotor-Auferregung     BOOL  ASG01  false      false
Taktart WR                WERT  ASG01  128        128
Fahrdrahtspannung         UU.U  ASG01  16.1       0.9
...
```

Figure 9: Examplary diagnostic code and corresponding system variables

again. These events are not documented as such, and we can only avoid negative influence on the training by removing the instances from the training set completely.

In the case of double repairs, we treat layovers caused by the same reasons that appear in a less than two weeks time as a single one, therefore assuming that the reasons to bring the train in were correct in the first place. Unfortunately, we can not prove if this assumption is always correct, but a discussion with the domain experts assured us that it is usually the case. This information will help us in the labelling process such as we can remove those instances in between two repairs from the training set for the model.

5. Feature generation

In this section we will discuss the process of selecting diagnostic codes to create features. In the steps of importing the original datafiles into an SQL database, the original structure of the data-log — to each event a small set of system variables, the so called surroundings, is attached — was separated into two independent data tables (Fig. 9). This is an advantage in creating useful features, because we can choose which values are used from both parts of the data separately.

In the following sections we will discuss how and why we selected a certain subset of diagnostic codes and system variables. The combination of both of them will then be turned into instances (one for each train for each day). The challenge here is, to select the features that contain the information we need, but also to keep the amount of features as low as possible, because otherwise our generated set of instances will become very large and problems may arise when trying to learn a model on them (RAM issues, long training times, etc.). We train the models on the TU Darmstadt server cluster[2], which offers more than 1150 nodes (Oct. 2015) for parallel computing. We only need single-core nodes for each model-training process, but we have 500 parametrizations for this process, so we can run a minimum of 20 processes in parallel. If we run more, the database becomes a bottleneck.

5.1 Selecting useful diagnostic codes

In order to select a good subset of diagnostic codes to turn into features, the following aspects have to be accounted for:

- The chosen subset has to contain the necessary information
- The chosen subset should be as small as possible, which will require less space and processing resources.

5.1.1 Naive approach

The naive approach to this problem is: Use all available attributes and — by sheer processing power — determine which is the optimal subset of these attributes. While this would be possible for the diagnostic codes alone (one diagnostic code is turned into three attributes/features), including the system variables (where one status variable may turn into hundreds of resulting features) would make this very hard to calculate, even with the TU Darmstadt server cluster.

A more feasible approach would incorporate pre-filtering of the diagnostic codes with external knowledge or based on other metrics.

5.1.2 Selection by univariate correlation analysis

One of the possibilites to achieve a pre-filtering, is correlation analysis. In this case, we analyzed the correlation of each attribute with the target class (*failure*). Unfortunately, the result of this analysis showed that the most positive or negative correlated attributes are just diagnostic messages that appear very frequently anyway. Most of them cover completely unrelated messages, for example ”Führerraum 1 ein” (Operating Stand 1 enabled). Therefore we can only partially use the results of this correlation analysis: We assure that the correlated messages that don’t contain trivialities are contained in our final selection set.

5.1.3 Selection by system

The final approach to diagnostic code selection takes a coarse pre-selection based on the system and subsystem. For example, the LZB failure is coupled to the LZB system and the diagnostic codes which belong to this system. Nevertheless, diagnostic messages from the ZSG-system could also be relevant, etc. In order to make an elaborate choice on which systems and subsystems might be relevant, an expert interview was conducted.

The result was as follows: Relevant systems include LZB and ZSG.
Relevant subsystems are *EVC*, *ATP*, *IDU*, *LZB*, *LZBE*, *TRAN*, *GSTM*, *STMh* and *STM*. Those systems contain a total of 2132 different types of diagnostic messages, which narrows down the initial selection to about one third (from 6909). Afterwards, this selection can be further reduced by removing diagnostic codes that never occur, which reduces this selection to 693 diagnostic codes. All the codes from the previous correlation analysis are contained in this selection.

2. Lichtenberg-Hochleistungsrechner http://www.hhlr.tu-darmstadt.de/hhlr/

5.2 Selecting useful system variables

In addition to the diagnostic codes, we select useful system variables to be incorporated in our featureset and hence the daily instances. The system variables contain system states and analogue measurements (e.g. temperatures and pressure values) and can therefore be helpful of describing the underlying physical aspects of a failure scenario.

The selection of useful system variables — especially the system states — is nontrivial as well. We chose a hand-selected number of variables for each of the problems. Unfortunately, analysis of the system variables showed that many of the recorded values are always ”0”, so we do not use them and this reduces the amount of selected variables to a few. In the case of the LZB failure, the observed variables where

- Sensor-Status
- Interne ATP-Meldung
- Interne LZB-Meldung

All of these variables are status-variables, meaning they do not contain specific measurements, but only numbers which represent a state of the system.

5.3 Variable types

In this section we describe the three basic types that system variables can have.

5.3.1 Status variables and diagnostic codes

Status variables are usually filled with a hexadecimal number of a system state. Since there is no coherent documentation on the variables and the possible states of them, we had to do a complete search for all occurrences of each system variable in order to determine all the possible states. For some variables, this can be several hundred. Boolean variables — having the states *true* and *false* — are treated as state variables.

The diagnostic codes themselves, since they come with a *duration* information, can also be treated as a state. The features generated thereof are then equally processed and result in three values for each code/variable:

1. *Total duration of code/status*: All durations summed up for the whole day
2. *Frequency*: How often one diagnostic code/status occurs during one day
3. *Average duration*: Total duration divided by the frequency
4. *Tf-idf*: Term frequency inverse document frequency. This measure was originally used in the domain of information retrieval (cf. Sparck Jones (1972)) as a relevance measure for terms in a corpus of documents. It combines the frequency and the inverse document frequency, such that terms (in our case: diagnostic codes) that appear rarely in few documents (days) can be discriminated from regular terms that appear often.

These attributes cover the primary properties of the appearance of diagnostic codes and states. Other statistical values might be useful, e.g., variance of the average duration. It is planned to conduct further experiments including other statistics in the future, however, we are confident that these statistics have the highest impact on the significance of the features.

5.3.2 Numeric variables

Numeric values occur in a wide range of applications, for example the measurement of temperature values. For these variables we use standard statistical measures:

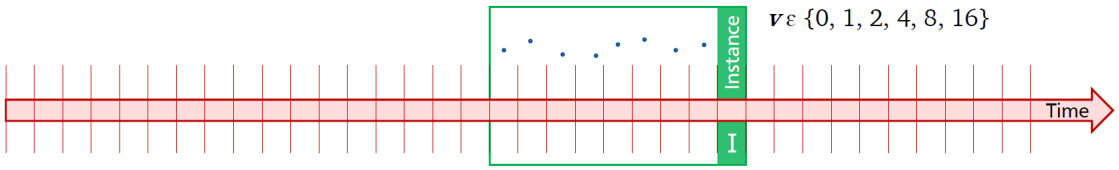

Figure 10: Instance creation: aggregating measurements of the previous v days into instance I

1. *Average*: Arithmetic mean of all recorded values in one day
2. *Maximum*: Maximum recorded value in one day
3. *Minimum*: Minimum recorded value in one day
4. *Variance*: Variance of all recorded values in one day

These attributes cover important properties of numerical values, more complex ones may be evaluated in later experiments.

5.3.3 TIME NORMALIZATION

Since a train does not have the same runtime each day, we scale the time-based values that are absolute (duration, occurrences) to the total uptime per day, in order to increase comparability between days. As a result we achieve frequency (occurrences per hour) as well as average duration per hour.

The combined selection of diagnostic codes and system variables described in this section and the attributes derived thereof result in a total of 2633 features per instance.

6. Labelling

In this section we will describe the labelling process with emphasis on the preprocessing steps.

In order to be able to train a model on the daily instances we created so far, we need to put a label to each instance that we want to train with. To do so, we discovered the ground truth in Sect. 4 and put together a useful set of features for our instances in Sect. 5. We will describe why large amounts of the data were not used for training and evaluation because of inconsistencies and information gaps.

6.1 Instance creation by aggregation

As proposed in related literature (Létourneau et al. (2005); Sipos et al.; Zaluski et al. (2011)), we will use a sliding window approach to label the instances (cf. Fig. 11). To calculate the instances, we will not only create a feature-vector A_t for a given day t, but instead calculate the trend leading towards this point in time.

For this, we use linear regression with the window size v for each day t such as to create a vector $I_{t,v} = linreg(A_{t-v}, ..., A_{t-1}, A_t)$, in order to represent the behaviour of the system in the last v days. The gradient of the linear regression is then used as the attribute.
Each of those vectors represents one instance, as can be seen in Fig. 10.

The labels are then assigned as follows:

Step 1: Label all instances as $warning = false$

Step 2: For each failure on train B and on day S, label the instances $B_{S-w}...B_S$ as $warning = true$

The value w represents the "warning epoch". The optimal value of w will be determined experimentally, and depends on the specific type of failure. The optimal value for v will also be determined experimentally. An example of labelling can be seen in Fig. 11.

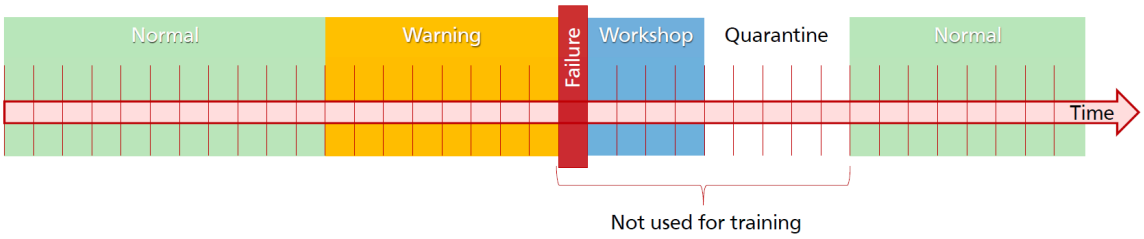

Figure 11: Label assignment

6.2 Quarantine area

Because of the nature of the sliding window, we need to assure that — right after a part has been repaired — we will not immediately create instances with $warning = false$.

For example, given a window size of $v = 8$ and a failure/repair on day F:
if we create $I_{(F+2),8}$ the window will date back to 6 days before the failure and incorporate the measurements from those days. The calculated features would be influenced by the behaviour before the maintenance.

Therefore we introduce the quarantine interval, also of length v. All instances in this interval may be affected by the failure and have to be treated accordingly, in our case removed. The quarantine interval prevents instances that are influenced by the effects of the failure, but labelled as $warning = false$ (depicted in Fig. 11).

6.3 Unnecessary layover area

In Sect. 4.4 we elaborated on how we detect unnecessary layovers. Apparently these result from values in the diagnostics system which caused it to issue a warning on the MFD. Thus, some sort of non-standard behaviour has been detected. Compared to our ground truth we can state that — although abnormal — the records do not correlate with the failure we are trying to detect. We do not want these to affect the training of the classifiers, so we create a buffer area around those occurrences. The buffer area affects all instances from $I_{t-v}...I_{t+v}$. The instances inside this area will not be used for training.

6.4 Removal of instances

As stated before, the diagnostic data we built the instances upon is not recorded continuously, but on an event-triggered basis. For example, data is not recorded when the train is switched off. To address this issue, the concept of *validity* was introduced. If no data was recorded on a given day, this day is regarded as *invalid*. The same applies, when no mileage was recorded on a day. It can happen that a train is switched on and records data, even when it does not actually drive. Most often this happens in situations where the train is moved to another rail, hence, we consider a mileage of less than 10 km per day as *invalid*, since driving less than 10 km definitely is no cargo delivery.

The last attribute that has an influence on the *validity* is the information, if a train was in the workshop at a given day. In workshop layovers, problem detection gear is usually attached and some diagnostic programs are executed, causing the train to emit more diagnostic messages than usual. In order to keep this artificially created information from influencing the process, workshop days are also handled as *invalid*.

In Fig. 12 the sequence of removal steps is displayed based on an example from the main air compressor problem. In the first part of the figure, the ground-truth (GT) resulting from the process of Sect. 4 is shown:

- The ”correct” line depicts events of correctly reported compressor failures. The compressor was tested and repaired.

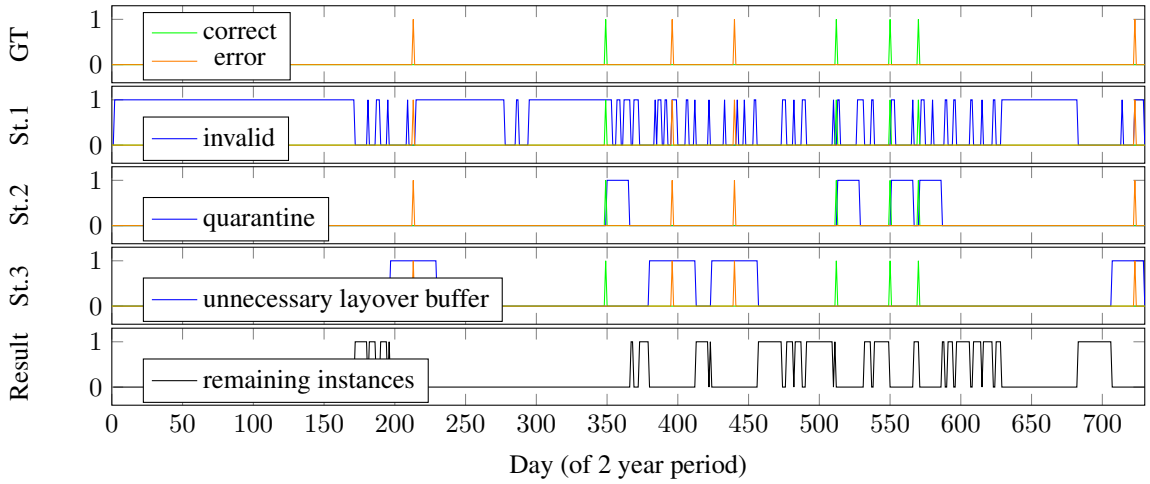

Figure 12: Stepwise removal of invalid and unreliable values

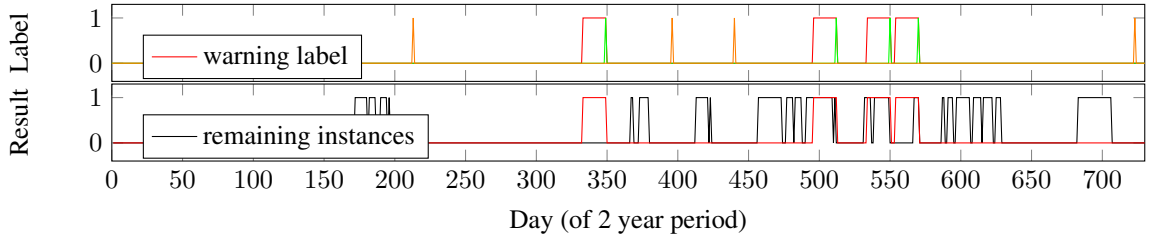

Figure 13: Remaining instances compared to positive labels

- The ”error” line shows unnecessarily reported failures. The test was negative and the compressor was not repaired.

During a period of 2 years, we calculate the conditions for an instance for each day. In steps 1-3 those criteria are displayed, the status being true (1) when the condition applies.

The first step of the removal process eliminates all the invalid instances (St.1). In the second step, we remove all instances that appear in the quarantine period defined in Sect. 6.2. Finally, we remove data in the unnecessary layover buffer area from Sect. 6.3 in step 3. This is done in order to eliminate all negative training influences those instances might have.

At the end of this process we are left with a significantly smaller number of instances, as can be seen in the *Result* column of Fig. 12. In comparison to the actual labels we assign to those instances, we can see in Fig. 13 that a significant number of the “warning=true” instances were removed during the process. The quality of those remaining instances with respect to our labelling is highly increased when employing these steps, since potentially problematic, useless or erroneous instances are completely removed.

7. Experiments

For the two types of tasks — prediction and unnecessary layover reduction — different experiments where conducted. They are essentially based on the same feature extraction and instance creation process, although the labelling might be slightly different depending on the task.

7.1 The prediction experiment

In the prediction task, the goal is to be able to classify instances from the *warning* period. For example, if we have learned a classifier with a 7-day *warning* period, we should get — in the real case — the first signs of warnings around 7 days before the failure. Sometimes we might get singular false warnings, but when warnings appear on consecutive days, this would indicate an impeding failure. For this task, the labelling as shown in Fig. 11 is used.

The complete dataset with 400 trains and the diagnostic data for 2013 and 2014 is processed as described. Unfortunately, conventional n-fold cross validation is not suitable for time-series data. In order to be able to achieve realistic results, it is important that the training and test-sets are independent in one of two ways:

- Time — a strict separation in time is chosen, for example 18 months of training data and the remaining 6 months as test data.
- Trains — as the trains are independent machines, we can separate our train/test dataset by trains. This means, we put the data from 90 % of trains into the training set, and keep the remaining 10 % as test data. We repeat this 10 times, so that each train is in the training set 9 times, and in the test set once.

We chose the second variant (split by trains) for our evaluation. It has the added benefit, that it could prove that failure behaviour is comparable between trains. With 6 different parameters for the warning period and 5 for the aggregation period, 30 combinations of each dataset were created.

The resulting datasets contain a very skewed class distribution, with more than 99 % of instances being labelled *normal*, and only few being labelled *warning*.

7.2 The unnecessary layover experiment

In this experiment, we want to know whether a classifier can differentiate between instances that belong to a layover with a repair (normal), or to an unnecessary layover. Only the instances of the day right before the failure are used for this experiment, the normal ones labelled *normal* and the unnecessary ones labelled as *fzfg* (Fehlzuführung). Since these instances represent one independent failure case each, we can rely on conventional 10-Fold cross validation for the evaluation.

Since about 30 % of failures were false warnings to begin with, the class distribution in this experiment is around 30 / 70 and not severely skewed.

7.3 Classifiers and further preprocessing

We used the WEKA Tools embedded in a java program to run the evaluation for both setups on. The following classifiers were chosen:

- JRip
- RandomForest
- BayesNet

As attribute selection methods, the following were chosen:

- CFSSubset
- ReliefF
- No attribute selection

This results in a total of 6 combinations of evaluating the datasets. The classifiers and attribute selection methods presented here were pre-selected based on the results. This way we have an ensemble tree method (RandomForest), a rule learner (JRip) and a bayesian classifier. We did not include a Support-Vector-Machine, because it would have needed significant performance tuning to be competitive, which can be a

demanding task on its own.
The chosen classifiers are robust against parametrisation mistakes and work well with the standard configuration provided by WEKA (which we used).

7.4 Results of initial experiments

Unfortunately, with either parametrization, none of the classifiers were able to predict the upcoming failures properly (prediction experiment). Also, distinction between correct and wrong failures is not possible (unnecessary layover experiment).

The instances were just too similar to each other to be distinguished.

7.5 The sampling experiment

As a further experiment investigating the shortcomings of our methodology, we tried some classification tasks which should be quite trivial, if the instances we created contain the information that is crucial to differentiate the failures from the normal operation. This experiment is based on the LZB Problem domain and the failures of the LZB System.

7.5.1 Experiment description

We sample a subset of instances as follows:

- We choose all real LZB failures that resulted in a repair/exchange of the system. This results in 163 failure incidents.
- For each failure that was found, we take one positive (warning) instance from the day before the failure took place.
- We then select one randomly chosen, negative (normal) instance for each failure instance.
- We ensure that the negative instances are not influenced by the failures by creating a buffering area (see Fig.14) from which no instances are chosen (40 days before and after each failure).

With this setup, we achieve big diversity between the normal and failure instances. If a real deterioration process is indicated by the data, the possibility for distinction should be increased by choosing the instances like this.

Furthermore we achieve evenly distributed classes, which is advantageous for learners. Especially with problems where one class is the minority by a large margin, some learners will not be able to learn the structure of the underlying class.

In order to avoid random anomalies in our result, we will repeat the sampling 20 times. In each of these iterations the positive instances will be equal, but the negative instances will be random. To put this in perspective, we have a total of 163 positive instances, and will chose 163 negative instances from a total of approximately 150000 available.

7.5.2 Parametrisation

The experiment setup regarding the classifiers and preprocessing is equal to the setup described in Sect.7.3.

For the instance creation phase, we used an aggregation window sizes of $v = \{1, 2, 4\}$ to create the instances as described in Sect.6.1.

For evaluation we used 5-fold Cross-Validation on each of the 20 sampling iterations. We can use cross-validation in this case, because the instances are independent of each other, unlike in the original prediction experiment.

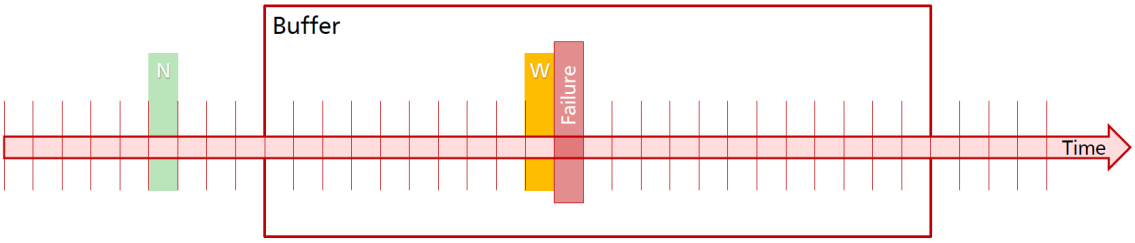

Figure 14: Buffered Sampling Experiment

Window	Classifier + Attr. Selection	AUC	Precision(w)	Recall(w)	Accuracy
v=1	RandomForest + ReliefF	**0.864**	0.813	**0.749**	**78.477**
v=4	RandomForest + ReliefF	0.86	0.81	0.735	77.96
v=2	RandomForest + ReliefF	0.855	**0.815**	0.735	78.024
v=4	RandomForest + CFSSubset	0.791	0.754	0.668	72.268
v=1	RandomForest	0.788	0.731	0.682	70.944
v=4	RandomForest	0.784	0.731	0.667	70.207
v=2	RandomForest	0.783	0.727	0.676	70.481
v=2	RandomForest + CFSSubset	0.771	0.732	0.653	70.466
v=1	RandomForest + CFSSubset	0.77	0.723	0.673	70.262
v=1	JRip + ReliefF	0.682	0.678	0.649	66.31
v=4	JRip + CFSSubset	0.667	0.671	0.643	65.126
v=2	JRip + ReliefF	0.666	0.674	0.617	64.824
v=4	BayesNet + CFSSubset	0.656	0.647	0.574	61.686
v=2	BayesNet + CFSSubset	0.651	0.639	0.541	60.748
v=1	JRip + CFSSubset	0.649	0.639	0.644	63.423
v=2	BayesNet	0.647	0.64	0.512	60.114
v=2	JRip + CFSSubset	0.647	0.653	0.628	63.691
v=4	BayesNet	0.647	0.637	0.553	60.388
v=4	JRip	0.646	0.645	0.641	63.385
v=2	JRip	0.645	0.642	0.641	62.974
...	(rest of results omitted)				

Table 1: Experiment Results ordered by AUC

7.6 Results of sampling experiment

When looking at the results of this experiment in Tab.1 we will focus especially on the recall and precision rates of the warning class. The results are ordered by Area under the ROC-Curve value, which is a combined measure that takes the true positives and false positives into account. We have limited the output to the top 20 results, as the remaining are of no specific interest.

As we can see in Tab.1, the RandomForest classifier generally performs high for this type of problem, regardless of the chosen attribute selection and window size. It achieves a precision of over 81.3 % and recalls more than 74 % of the instances.

The Ripper and Bayesian Network implementations of WEKA also perform well, although significantly below the RandomForest. In general, varying the window size has no significant impact. These results may vary when choosing substantially larger window sizes.

Although the results seem promising, there is one aspect we have to keep in mind: These results were conducted based on equally spread class distribution via 5-fold Cross Validation, meaning the classifier was trained and evaluated on this even distribution.

In a real scenario, we would not have equally distributed data which we apply the classification upon. We have to do further experiments in order to be able to estimate the performance on a real dataset.

8. Conclusion

In this report we have described various processes, which will help us solve the problems of failure prediction and reduction of unnecessary layovers:

- We have established a process to determine the ground truth for different failure types in order to be able to turn it into a supervised learning problem, which deals with unreliable data and covers aspects such as unnecessary layovers or double repairs.
- We have derived useful features from the diagnostic data that is recorded in the trains.
- We have deducted a labelling schema in order to be able to solve the tasks given by the problems at hand.
- We have performed preliminary experiments.

Although the results of the preliminary experiments seem promising, we can not confidently state that the methods as described will prove sufficient for the given problem types. In order to do so, we must further investigate the behaviour of our datasets given a more realistic situation, especially with respect to skewed class distributions and the effects they have on learning algorithms. This has to be analysed carefully to establish a solid and working method.

9. Future Work

For future work, we will look into the following topics:

- Investigate the skewed class distribution issue and develop an approach that helps classifiers to be trained and evaluated on such data.
- Analyse the instances of the sampling experiment, e.g. by clustering or statistical tools
- Switch from instances that cover days to instances that cover tours.
- Apply a different method to handling the warning period, for example a multi-instance approach.
- Use more fine-grained temporal binning. The original assumption of 1 complete day per instance might be too coarse.
- Gather more precise information to enhance ground truth. With the incorporation of the CDIF database, we might be able to improve the knowledgebase.

List of Figures

List of Tables

References

Sebastian Kauschke, Immanuel Schweizer, Michael Fiebrig, and Frederik Janssen. Learning to predict component failures in trains. In Thomas Seidl, Marwan Hassani, and Christian Beecks, editors, *Proceedings of the 16th LWA Workshops: KDML, IR and FGWM*, pages 71–82, Aachen, Germany, 2014. CEUR Workshop Proceedings. URL `http://ceur-ws.org/Vol-1226/paper13.pdf`. urn:nbn:de:0074-1226-4.

Silvain Létourneau, Chunsheng Yang, Chris Drummond, Elizabeth Scarlett, Julio Valdes, and Marvin Zaluski. A domain independent data mining methodology for prognostics. In *Essential technologies for successful prognostics : proceedings of the 59th Meeting of the Society for Machinery Failure Prevention Technology*, 2005.

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. ISSN 0268-3768. doi: 10.1007/s00170-009-2482-0. URL `http://dx.doi.org/10.1007/s00170-009-2482-0`.

Ruben Sipos, Dmitriy Fradkin, Fabian Moerchen, and Zhuang Wang. Log-based predictive maintenance. In *KDD '14 Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*.

Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

Marvin Zaluski, Silvain Létourneau, Jeff Bird, and Chunsheng Yang. Developing data mining-based prognostic models for cf-18 aircraft. In *Journal of Engineering for Gas Turbines and Power*, volume 133, 2011.