Predictive Failure Data Analysis in the AddTrack System

Simon Johansson
M.Sc. Scientific Computing

Addiva AB

December 18, 2020

Abstract

This report uses event activations within the AddTrack system in order to predict train failures. The results show a difficulty in performing such failure predictions with any sufficient precision, since the best precision lies in the range of 70% to 75%. It is concluded that more data, as well as a better data quality, should enable improved predictions.

1 Introduction

The AddTrack system is a railway operational data system owned, developed and maintained by Addiva AB in Västerås, Sweden. The system provides event data from trains around the world. The purpose of this document is to investigate how the data in the AddTrack system can be used to predict failures that occur on the trains. The only prior work that has been done to this end is the master's thesis [2], where machine learning was used on the available event data. However, the implementation in [2] shows abysmal results when it comes to validation accuracy.

This document will revisit the master's thesis [2], see if its findings can be improved, and suggest improvements of the data availability in the AddTrack system.

It is recommended to have read the master's thesis report, since some details are omitted here.

2 Data Structure

Every train in the AddTrack system contains units which continuously produce event data. An event, in this context, can either be On or Off at any given point in time. It can, for example, be the event of a door being open or closed, or the event of a battery being below its minimum voltage level or not. However, the system itself does not keep track of whether an event is active at a particular point in time. Instead, every time a train dumps its data in the AddTrack system, it dumps the list of new event activations since its last dump, for some trains also including the list of de-activations. Thus, the only data that is always available is the count of event activations per time period. This is the data used in [2], as well as in this report.

The event data used, both in this document and in [2], is event data in the AddTrack system from the years 2018 and 2019. In addition, there is also a set of failures data provided not through the AddTrack system but instead as handwritten reports from the same time period. These two sets of data are used trying to predict failures from the set of events.

3 Revising the Master's Thesis Results

3.1 Setup

To begin with, an effort was made to replicate the findings of the master's thesis [2]. To do this, the following steps were taken:

1. Database preprocessing was performed, including the split of event data into one section for electric trains and another for diesel trains, as in [2]. This is logical since the set of events for electric trains is not the same as the set of events for diesel trains.
2. Code was implemented to generate the inputs needed by the ML algorithm, trying to replicate the logic described in [2].
3. The same ML implementation was used as in appendix A of [2].
4. A table of accuracy outputs was generated in order to compare with table 4 in [2].

In this context, step number 2 is elaborated below.

3.2 Generation of Inputs to the ML Algorithm

As in [2], only failures data for the compressor and the pantograph were studied, since the number of failures was deemed to few in the other categories. To predict failures, for example on the compressor, the data was divided into time periods, where each time period is a number of days, and where each day is split into a number of time frames. For each time frame, the number of event activations was aggregated for each type of event.

In order to generate data that could be used for training and testing, two kinds of data periods were generated:

- *Label 0*: Time periods containing no failure.
- *Label 1*: Time periods followed by a failure.[1]

This was done in the same manner as in [2]. Thus, a time period to be included in the *Label 0* set was generated starting at January 1st 2018, with a given period length, if there was no failure of the given type during that time period. The next time period to be included in *Label 0* was generated starting a week later, if there was no failure of the given type during that time period, etc. Time periods to be included in the *Label 1* set, however, were generated by going through all the failures of the given type, setting the date of the failure to the day before the failure report (see [2]), assuming a two day "prediction window" prior to the failure, letting the period created end at this point in time, and then, using Synthetic Minority Oversampling Technique (SMOTE), generate several time periods added to the *Label 1* set for each reported failure. The number of such generated time periods for each failure then depended on how many days backward in time that were used while generating periods with SMOTE. Also, to get the same behaviour as in [2], the set of *Label 1* periods was additionally replicated to get roughly the same number of time periods in the *Label 1* set as in the *Label 0* set.

Finally, the first 80 % of the time period from January 1st 2018 to December 31st 2019 was used to generate training data, and the last 20 % of the same time period was used to generate validation data.

3.3 Run Comparison with Master's Thesis

Runs where made, with similar parameter values as in [2], and the results can be seen in table 1, see below.

[1] There is a small chance for an overlap here, but this chance is negligible and ignored.

T_1 Acc	T_2 Acc	T_3 Acc	T_4 Acc	V_1 Acc	V_2 Acc	V_3 Acc	V_4 Acc	Days Period Length	Days Preceeding	Failure Type	Train Type
78.31	97.77	97.93	98.18	93.70	93.70	93.70	93.76	14	4	Pantograph	Electric
87.21	98.27	98.55	98.48	93.76	22.74	93.82	93.70	14	2	Pantograph	Electric
87.90	97.91	98.61	98.59	93.95	93.89	93.58	93.70	28	2	Pantograph	Electric
78.63	97.36	97.89	97.90	93.89	94.14	94.14	93.89	28	4	Pantograph	Electric
91.64	99.16	98.95	99.08	94.52	94.52	94.52	94.29	14	2	Compressor	Diesel
85.54	98.70	98.70	98.70	94.52	94.74	94.52	94.52	14	4	Compressor	Diesel
93.57	99.44	99.35	99.31	95.38	93.34	94.33	93.40	14	4	Compressor	Electric

Table 1: Accuracy in % for runs comparable with table 4 in [2], where T_j *Acc* refers to training accuracy during epoch j, and V_j *Acc* refers to validation accuracy during epoch j. NOTE: The run with a days period length of 14, days preceeding of 4, failure type compressor and train type electric could not be performed since the matrix in question became too large to allocate in RAM memory.

These results can be compared with table 4 in [2], although that table has some problems. For example, there are 3 lines in that table that have the parameters *(Days preceeding = 4, Failure type = 5 (pantograph), Train type = R (electric train))*. The author probably meant to include other parameters as well but made a mistake. Anyhow, the most important take is that the accuracy values in table 1 are radically better than those in table 4 in [2]. The only exception to this is the validation accuracy of the case *(Days Period Length = 14, Days Preceeding = 2, Failure Type = Pantograph, Train Type = Electric)* during the 2nd epoch validation, which is only 22 %. Other than this strange value (which probably is due to the (stochastic) dropout), the accuracy values in table 1 are between 93 % and 96 % during validation, while the validation accuracy values in table 4 in [2] ranges from 4 % to 49 %.

This invalidates the conclusion in [2] that the algorithm is unable to generalize from training to validation. Indeed, after studying the code used in the master's thesis, as well as comparing with the above results, it was concluded that the master's thesis code has an implementation error. To be more precise, it does not correctly map events to indices in an event data array, which needs to be done correctly. To exemplify, let's say that event e_1 is activated 2 times in period 4 and 3 times in period 5: The code in [2] is unable to see that this has happened, since it doesn't map indices correctly, and is therefore unable to see that it is the same event that is activated in the two different periods. This is more than problematic and means that the ML algorithm cannot learn, since the data it receives is not internally consistent.

Anyhow, table 1 shows the results of the new implementation that works, and that is able to generalize. The question is now how we can build on these results in a productive way to generate predictions.

3.4 True Positives, True Negatives, False Positives and False Negatives

One potential problem with the above implementation is that it uses the same loss function that is used in [2], namely the (binary) accuracy function, which checks the prediction for each case and compares it with the actual value, counting the number of correct predictions, and dividing with the total number of cases. In reality, this means that the loss function only cares about true/false predictions, not about true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). This is problematic since an FP is very costly, because it means sending out a team for fixing a train... where there is actually nothing wrong with the train. An FN, on the other hand, is not as costly, it just means sending out a team after the failure, just as is already happening today.

Now, although the best loss function is not currently being used, it would still be interesting to show the TP, TN, FP and FN values for the runs from the previous section, see table 2 below.

From looking at the table, we see that the algorithm cares a lot about minimizing the number of FN values, not so much about minimizing the number of FP values. This is a bad priority, which we will seek to remedy in the next section, along with other considerations to improve the results.

4 Loss Function and Up/Down Sampling

In the previous section it was shown that the master's thesis [2] has implementation errors which are remedied in this report. It was also shown that, in most of the given cases, a validation accuracy of 93 % was reachable by the algorithm. Finally, it was declared that (binary) accuracy might be an insufficient measure. Other problems, that were not discussed, can be summarized as follows:

1. There is a lot of data, both when it comes to RAM memory use and disc use.
2. The periods in *Label 0* are overlapping, e.g. a new such period starts every week, while the period length is two to four weeks long, which is actually a form of oversampling for the bigger of the two data sets.

V Acc (%)	FP	FN	TP	TN	Precision (%)	Period Length (Days)	Days Preceeding	Failure Type	Train Type
93.76	102	2	136	1427	57.14	14	4	Pantograph	Electric
93.70	102	3	135	1427	56.96	14	2	Pantograph	Electric
93.70	96	6	132	1386	57.89	28	2	Pantograph	Electric
93.89	93	6	132	1389	58.67	28	4	Pantograph	Electric
94.29	76	0	127	1129	62.56	14	2	Compressor	Diesel
94.52	73	0	127	1132	63.50	14	4	Compressor	Diesel
93.40	106	1	66	1449	38.37	14	4	Compressor	Electric

Table 2: Validation accuracy (V Acc), false positives (FP), false negatives (FN), true positives (TP), true negatives (TN), and precision (defined below, given in %) for the *last* epoch (epoch 4) in the runs comparable with table 4 in [2].

3. The periods in *Label 1* are oversampled using SMOTE, and then replicated, which means that the same data is processed again and again, which tends to lead to overfitting.

4. The replication mentioned in 3 is not exact. If replication is performed, then maybe one should try to make the two sets the same size, rather than similar (although this may be of minor importance).

This section is about trying to improve on the above problems.

4.1 Switching Loss Function

It is reasonable that the loss function is consistent with the most sought for measure. Thus, we would want a loss function that strives to minimize the number of FP values in some sense. One way to do this is to write a custom loss function that optimizes for precision (see below), and use it in the model (replacing the binary cross-entropy function used until now).

In order to use a function as a loss function, it is required that its gradients can be computed at all times and are different from the zero vector. Otherwise, the inputs would not be useful for the optimization algorithm.

The standpoint taken in this report is that false positives are costly, and more important to minimize than false negatives. Thus, what we want is a

loss function which optimizes on the precision/recall scale, where precision is premium. Before this is done, some definitions are provided:

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

This means that precision is the number of true positives divided by the number of predicted positives, and that recall is the number of true positives divided by the number of actual positives. Note that precision and recall have a weak inverse relation: If the number of FP values increase, then, ceteris parabus, the number of FN values either stays the same or decreases. This is called a precision/recall tradeoff. See for example [1].

In general, one would then want to balance precision and recall. One way to do this is to construct the weighted precision recall measure:

$$W_{PR} = \alpha P + (1 - \alpha)R, \tag{3}$$

where P is precision, R is recall, and $\alpha \in (0, 1)$. If α is close to zero, then we say that false positives are of no greater concern, and if α is close to one then we say that we say that false positives are of a great concern. Also, in the first case we will allow relatively many positive predictions, and in the second case relatively few. Thus, for the current problem we would want α to be relatively close to one.

To construct a loss function from (3) one can simply use

$$W_{PR}^{loss} = 1 - W_{PR}, \tag{4}$$

since a loss function is always supposed to be minimized. It is worth noting that there will be a difference between implementation and theory since the optimizer, RMSProp, will utilize floating point values between zero and one when it iterates. This means that the input array of predictions sent to the loss function will be between zero and one rather than actual zeros and ones. Thus, in (4) we will use the following:

$$TP + FN = \sum y_{true}, \tag{5}$$

$$TP + FP \approx \sum y_{pred}, \tag{6}$$

$$TP \approx \sum y_{true} y_{pred}, \tag{7}$$

where y_{true} are the true values (zero or one) defined for each period, and where $y_{pred} \in [0, 1]$ are the predictions.

4.2 Down/up sampling

The following types of sampling is used in the master's thesis, as well as for the runs referenced in section 3.3:

- The *Label 0* set is "up sampled" (in some sense), by creating overlapping time periods,
- The *Label 1* set is up sampled by 1) SMOTE, and by 2) replication, to reach approximately the same size as the *Label 0* dataset.

As part of the experimentation for this report, it was tried to construct the *Label 0* set so that there would be no overlaps between its members. This did not turn out well, however, with accuracy levels falling sharply, so the old implementation was kept in this regard. Instead, the replication used when constructing *Label 1* was dropped, and a down sampling procedure called Agglomerative Clustering was applied to the *Label 0* set to make it the same size as the *Label 1* set.

4.2.1 Agglomerative Clustering

Agglomerative Clustering is a flexible method to divide a large dataset into a set of clusters. In the current implementation, this means starting with the dataset described as *Label 0* in section 3.2, and then, using a suitable measure, splitting it into "adjacency clusters" (where each period itself is considered a cluster). To be more precise, the following is done:

1. The set of all time frames in a single period is aggregated into a single array of event activations (i.e. this array contains a single value for each event, which corresponds to the total number of event activations for that event during the given time period).
2. The euclidean distance is defined in the normal manner between the event activation arrays of every pair of periods.
3. A bottom-up approach is used to iteratively merge nearby clusters with one another, recursively forming clusters containing clusters until a single cluster is formed, which covers the entire dataset.
4. The cluster of all clusters is finally split on a suitable hierarchical level so that the sought for number of clusters is generated (discarding all clusters above this level).
5. From each of the selected clusters (which now are as many as was requested), the first period is added to a new modified *Label 0* set.

6. The new modified *Label 0* set is sent to the ML algorithm instead of the original *Label 0* set.

To implement the above clustering method, scikit-learn is used, which has a built in Agglomerative Clustering implementation (using parameters: affinity='euclidean', linkage='ward'). See [3] for the sci-kit learn documentation.

4.3 Results

The configuration described in sections 4.1 and 4.2 is run using Compressor as failure type, and Electric as train type, where comparison is made between the Binary Cross-Entropy function, and the weighted precision-recall function, see tables 3 and 4 below.

Acc	FP	FN	TP	TN	Precision	Period Length	Days Preceeding	Failure Type	Train Type	Loss Function
93.34	108	0	67	1447	38.29	14	4	Compressor	Electric	Binary Cross-Entropy
94.33	90	2	65	1465	41.94	14	4	Compressor	Electric	Weighted Precision Recall 80/20
94.33	89	3	64	1466	41.83	14	4	Compressor	Electric	Weighted Precision Recall 95/5

Table 3: Comparison of loss function implementations, last epoch out of 10. Values are for validation.

The following can be observed: 1) The binary cross-entropy loss function leads to an ML implementation that learns faster, since it has 42% in validation precision already after the first epoch, but then has a tendency to start falling, most probably due to a combination of overfitting and the fact that it optimizes for accuracy rather than precision. 2) The weighted precision-recall implementation with 80% weight on precision and 20% weight on recall starts out poorly, but becomes quite stable at 41.94% from the 7:th epoch and onwards. 3) Both the weighted precision-recall implementations tends to fewer predicted positives than the binary cross-entropy implementation. 4) The weighted precision-recall with an 95/5 split tends to even fewer predicted positives than the 80/20 split (see for example epochs 1 and 3).

P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	Failure Type	Train Type	Loss Function
42.21	41.83	41.83	41.40	41.56	38.29	42.58	41.36	38.73	38.29	Compressor	Electric	Binary Cross-Entropy
4.13	4.13	4.13	4.13	32.66	42.31	41.94	41.94	41.94	41.94	Compressor	Electric	Weighted Precision Recall 80/20
-	50.00	-	48.98	42.57	41.83	41.83	41.94	42.21	41.83	Compressor	Electric	Weighted Precision Recall 95/5

Table 4: Comparison of precision values for different loss function implementations for 10 epochs. P_j is measured validation precision at epoch j. A dash means that precision could not be measured since there were no predicted positives.

This argues that weighted precision-recall might be better in the long run (i.e. when running many epochs) than binary cross-entropy, so the precision-recall implementation is kept while the binary cross-entropy implementation is dropped. Further experiments was also carried out using pantograph failures (as well as compressor failures) for electric trains. In this case, there was no use of recurrent drop out (for the LSTM weights), with the thought that too much long temporal features could become lost with its use. Similar runs where made, see tables 5 and 6 below, but now also including a weighted precision-recall of 99 and 1 percent, respectively.

The results are interesting: First of all, when using an 80/20 split or an 95/5 split, compressor failures are predicted with about 42% certainty (for electric trains), while pantograph failures are predicted with about 57% certainty. Both values are, off course, rather bad from a predictive standpoint, but if we compare this with the runs using a 99/1 split, then we see values in the range 57% to 75% for compressor failures, and 57% to 59% for pantograph failures, where the first case actually is promising. It's promising because it indicates that the weighted precision-recall loss function might have some usefulness - that higher precision might actually become possible with a better parameter use, a bigger validation set, the right techniques etc, so that it can become useful in real-life.

To be fair, we need to look at not just to the precision of the predictions, but also to the actual number of (positive) predictions. As can be seen in table 5, the number of predicted compressor failures during validation, for the 99/1

Acc	FP	FN	TP	TN	Precision	Failure Type	Train Type	Precision/ Recalll
94.39	90	1	66	1465	42.31	Compressor	Electric	80/20
93.70	102	3	135	1427	56.96	Pantograph	Electric	80/20
94.39	89	2	65	1466	42.21	Compressor	Electric	95/5
93.70	102	3	135	1427	56.96	Pantograph	Electric	95/5
95.93	3	63	4	1552	57.14	Compressor	Electric	99/1
91.72	0	138	0	1529	-	Pantograph	Electric	99/1

Table 5: Comparison of weighted precision/recall implementations, last epoch out of 10. Values are for validation. A dash means that precision could not be measured since there were no predicted positives.

P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	Failure Type	Train Type	Precision/ Recall
4.32	42.18	42.21	41.94	42.31	41.94	41.94	41.94	41.83	42.31	Compressor	Electric	80/20
8.28	8.28	8.28	56.96	56.96	56.96	56.96	56.96	56.96	56.96	Pantograph	Electric	80/20
-	61.90	41.83	42.21	41.83	41.83	42.67	44.68	42.21	42.21	Compressor	Electric	95/5
56.96	56.96	56.96	56.96	56.96	56.96	56.96	56.96	56.96	56.96	Pantograph	Electric	95/5
-	-	75.00	66.67	60.00	-	-	-	57.14	57.14	Compressor	Electric	99/1
56.96	56.96	-	59.09	-	-	-	-	-	-	Pantograph	Electric	99/1

Table 6: Comparisons for 10 epochs using different weighted precision-recall values. P_j is measured validation precision at epoch j. A dash means that precision could not be measured since there were no predicted positives.

split, were 7 (true positives plus false positives), and 4 of them were predicted (true positives). It can be concluded that the higher predictive rate has come with the cost of fewer predicted failures.

5 Further Experimentation with Set Size and SMOTE

The most promising results up until now comes from the use of the weighted precision-recall function with a 99/1 split. The problem is that this yields too few failure predictions, and therefore the precision values become either non-

existent or based on very few positive predictions, which itself is unreliable.

5.1 Increased Test Set

One approach to remedy this might be to increase the test set. This is done by splitting the data set at 50% over the given time period. In other words, the training data now consists of the time period covering the year of 2018, and the validation data the year of 2019. Off course, this has the drawback of yielding a smaller training set, but is still worth investigating.

Assuming this change in time periods, the weighted precision-recall with a 99/1 split is used, and the results can be seen in tables 7 and 8.

V Acc	FP	FN	TP	TN	Precision	Failure Type	Train Type
98.38	0	67	0	4060	-	Compressor	Electric
96.74	47	87	51	3926	52.04	Pantograph	Electric
96.80	104	0	127	3024	54.98	Compressor	Diesel

Table 7: Results for a 50% split between the time period for the training data and the validation data, last epoch out of 10. Values are for validation. A dash means that precision could not be measured since there were no predicted positives.

P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	Failure Type	Train Type
-	-	46.15	-	66.67	-	-	66.67	-	-	Compressor	Electric
50.19	52.42	49.45	49.45	49.45	49.45	49.45	49.27	49.27	52.04	Pantograph	Electric
58.62	55.56	54.98	54.98	54.98	54.98	54.74	54.74	54.74	54.98	Compressor	Diesel

Table 8: Comparisons for 10 epochs using a 50% split between the time period for the training data and the validation data. P_j is measured validation precision at epoch j. A dash means that precision could not be measured since there were no predicted positives.

Note that we include results for compressor failures on diesel trains, with precision values between 54% and 59%, which are not radically different than the other results. The results are also similar to earlier which means that any gains from an increased test set has been roughly canceled by the decreased training set.

5.2 Removal of SMOTE

Another question is how important the use of SMOTE is for the results. To test this, the use of SMOTE is removed, which means fewer (and bigger) cluster groups in the *Label 0* set to keep the sets of *Label 0* and *Label 1* at the same size. This also leads to very small training sets, requiring many more epochs to yield any reasonable results, and raises the question of convergence, which, up until now, has been ignored. To tackle these issues a learning rate scheduler is introduced, which decreases the learning rate over the long run, as well as an early stopping callback that checks for improvements and stops if the results are no longer improving.

The learning rate scheduler consists of a function which takes the default learning rate r as well as the epoch number j and yields a modified learning rate:

$$r_{mod} = \begin{cases} r & \text{if } j \leq 10, \\ \frac{r}{1+j-10} & \text{otherwise,} \end{cases} \tag{8}$$

which means that the learning rate used remains constant the first 10 epochs and then starts decreasing. The early stopping mechanism introduced checks the value of the loss function after each epoch and if the value of the loss function has not decreased during the last 5 epochs it stops the execution of the ML algorithm. Finally, to create results with a smaller variation over the epochs (with the weakness of increased overfitting) the drop out value is set to zero. This is primarily motivated by the fact that the training data set now is so small that dropping nodes risks making the results useless.

The results using the above changes, with a time split of 80% training data, 20% validation data, is shown for the last epoch in each case, see table 9.

V Acc	FP	FN	TP	TN	Precision	Failure Type	Train Type
91.72	0	138	0	1529	-	Pantograph	Electric
90.47	0	127	0	1205	-	Compressor	Diesel
96.12	3	60	7	1552	70.00	Compressor	Electric

Table 9: Results for ML runs using no SMOTE, the last epoch in each case. Values are for validation. A dash means that precision could not be measured since there were no predicted positives.

There are two interesting facts about these results: 1) There are (again) very few predicted positives, yielding no precision values in the cases pantograph for electric trains and compressor for diesel trains. 2) The 10 predicted positives in the case of the compressor for electric trains have 7 true predictions, which

yields a precision of 70%. Thus, yet again we see that there is cause for being cautiously optimistic but, in general, the amount of data seems insufficient.

6 Data Availability

The data used in the analysis is the number of activations per time period (or, specifically, per time frame), as well as handwritten failure reports. This limits the analysis. The question is, what could be done in order to improve the data? This could be either through providing more data of the same kind, presumably yielding a higher number of failure predictions, which should improve the precision results. It could also be done in other ways, by improving the data itself, or by providing other kinds of data, enabling other types of analysis.

6.1 Data Limitations

The following data problems exist relating to the above analysis:

- The failures data is imprecise, i.e. we have the time of a failure report, but not the time of the first observation of a failure. This increases the uncertainty of the actual time of the failure incident.
- The failures data does not include information about if or when a train was pulled off the track for reparation purposes, neither does it include information about when a failure was resolved. This means that there is the possibility of irrelevant data from when the train was off the track, as well as the increased probability of when events could have happened after a failure, being the consequence of the failure, and still be part of a period placed in the *Label 0* category.

Finally, data problems exist in the sense that they prevent an improvement in the analysis:

- There is no continuous information about the state of the trains in the AddTrack system, for example when it comes to sensor data, although this information is available on the trains themselves. This is because events happen irregularly, and because AddTrack doesn't receive information between events. If this information was available in the AddTrack system, it would be easier to monitor the trains, relating this data to the events and/or to the failures.

- Event de-activations are not always available in the data. This means that the AddTrack system cannot deduce during what time periods a certain event was active or not. Even if 100 % reliability is impossible, since packages can be lost during network communication, there should still be ways to improve the system so that we, in general, could deduce this. This would then be a new kind of data, it would enable a different kind of analysis, where active events could be related to failures rather than connecting the number of event activations to the failures.

6.2 Data Model

The question is to what extent the above problems can be resolved. In order to answer this question, a data model has been provided, see figure 1.

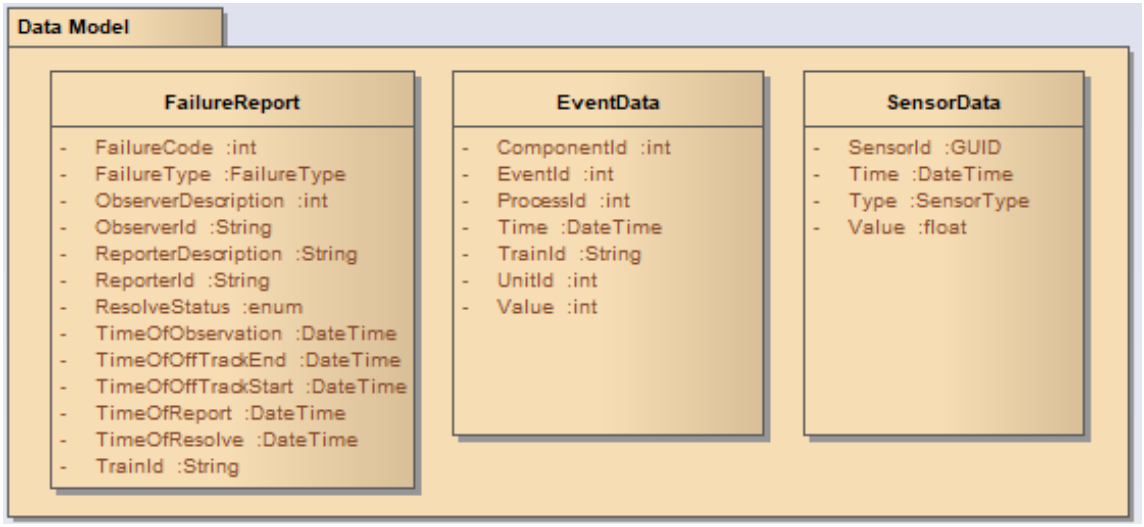

Figure 1: Data model for AddTrack data from the perspective of improving the analysis.

The model illustrates that the most interesting data generated from the trains are 1) failures, 2) events, and 3) sensors data. Failures are adverse problems that either prohibit the train from operating properly, or at all. Events are signals that describe what is happening on the trains, and so is sensor data. If all these three are made properly available, then this would increase the probability of performing both predictions and failure diagnostics. The data of today is too poor to be of greater use, even though there is certainly some merit to it, as has been shown in the previous sections.

An important point of the model is the time of first observation of a fail-

ure, which is something that (usually) the driver would initiate, giving a first rudimentary description of a problem, initiating a report. This could be done through a user interface provided to the driver. Later, the same or another person, for example a technician, could add more information to the initiated failure report, including revising or setting a failure code and failure type (these could, tentatively, have been set by the driver and then modified by the technician). There would also be the possibility to note off-track time, which would signal data usually irrelevant to the analysis. Finally, when the failure has been resolved, this would be noted as well, and at all times there would be a resolve status of the failure, making it clear if a problematic state persists.

Event data, on the other hand, needs to include data that it already does, such as event id and process id, but also unit id and component id, which describes from where, i.e. from which exact component, the signal comes. It is also important to note that the event would have an event value (for example '1' meaning active, and '0' meaning inactive), and that - no matter if the event is set to either of its available states - this would be sent as a signal to the AddTrack system. This means that both event activations and de-activations would be sent to the AddTrack system, so that the state of the event can always be tracked.

Finally, to keep track of continuous data from sensors, the train would provide real time data for the current values of all its sensors. This could, for example, be achieved by the dumping or streaming of events data at every X minutes, depending on the needs of granularity. This would happen independently of the event data signals, which in turn should be sent to the AddTrack system immediately (if possible) when an event changes state.

7 Conclusions

This report concludes that the code implementation used by the master's thesis [2] suffers from a problematic implementation error which invalidates the data sent to the machine learning algorithm. This is shown by comparing the first results of this report with [2], as well as by analyzing the code used by the master's thesis. Instead this report concludes that the event data activations can be used to perform predictions of an accuracy of 93%, and above. However, it also concludes that it is difficult, with the current data, which contains relatively few failures, to have a sufficient precision in the predictions for this to have any real-world use. The machine learning algorithm can be tweaked, and different methods can be used to pre-process the data, but in the end the small number of failures makes high precision difficult to achieve.

The best results are shown when utilizing the weighted precision-recall loss function, with a 99/1 split, where the precision at times reaches 70% and above.

It is conceivable that a different loss function, or a different general approach to the machine learning problem, including a different optimizer, could render better results but that needs to be investigated further.

The main obstacle to better precision is still poor data. Examples of needed data includes the time of observation of a failure, resolve status of a failure, event de-activation signals, and continuously streamed sensor data. More data of the same kind as today could still be used, however, and might lead to an increased predictive ability of the model. But this would have its limitations. In order to truly take control of the data, to improve the ability to make machine learning predictions (as well as diagnostics) there would need to be some changes to the data availability, as described above.

References

[1] Géron A, *Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems*, O'Reilly, 2nd Edition, 2019

[2] Hric J, *Event Based Predictive Failure Data Analysis of Railway Operational Data*, Thesis for the Degree of Master of Science in Computer Science, Mälardalen University, School of Innovation Design and Engineering, Västerås, Sweden, 2020

[3] Scik-Kit documentation of Agglomerative Clustering `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html`, accessed December 4, 2020