10º CRP CONGRESSO RODOFERROVIÁRIO PORTUGUÊS

A Deep Learning Approach for Predictive Maintenance: predicting future event faults within the TimeKeeper by EFACEC system

José Manuel Sousa[1], Ricardo Ribeiro[2]

[1]EFACEC Engenharia e Sistemas S.A., Via Francisco Sá Carneiro, Apartado 3078 · 4471-907 Moreira da Maia, Portugal

email: jmsousa@efacec.com

[2]Faculdade de Ciências, Universidade do Porto, Rua do Campo Alegre, 4169-007 Porto, Portugal

Abstract

Predictive Maintenance can be viewed as a technique for monitoring the operating condition of a system, providing data that enables the maximization of the time intervals between repairs and minimizes the unscheduled stoppage time due to machine failures. With the rise of digitization, Predictive Maintenance is moving from Condition-based Maintenance to Analytics-and IoT-based Predictive Maintenance, due to the explosion of data availability and the significant reduction of embedded systems cost. This paper investigates the use of Machine Learning in the context of Predictive Maintenance, more specifically, focusing on supervised learning for solving regression problems, using Neural Networks (Deep Learning) to estimate future events within the TimeKeeper by EFACEC system.

TimeKeeper by EFACEC helps operators to run buses and light rail vehicles on time and to optimize network operation, providing services that enable fleet and driver operational management, real-time service optimization and passenger information. Within this system, a myriad of events is captured and stored in real-time, which can be described as a set of attributes: type of event, level, value, timestamp, etc. These different classes of past events can be used to detect and predict future occurrences of the same class of events. In this context, special artificial neural networks, Long Short-Term Memory networks, were used to achieve data-driven models that make use of this historical data, to predict future events for different classes of events. The Area Under the Curve metric is used to compare models, and the average value was 0.9456, which is an exceptionally high value, proving the validity of the approach reported in this paper.

Keywords: Predictive Maintenance; Machine Learning; Neural Networks; Long Short-Term Memory.

1. Introduction

In a world where the rise of digitization enables a growing number of different possibilities, Predictive Maintenance (PdM) arises as being one of them. In fact, the availability of large amounts of data (historical or in real-time) using different types of sensors, changes and faults can be detected, and algorithms can be applied to convert the noted changes into actions in a way that machines and humans are able to take preventive actions to avoid future faults. In this particular context, an article from IIOT World (Fogoros 2018), reports that sensors and Machine Learning (ML) are the two main trends in the PdM space right now.

PdM has many different definitions. (Chiu, Cheng, and Huang 2017) state that the primary goal of PdM is to save money and improve equipment reliability. (Amruthnath and Gupta 2018) defend that its main purpose is to reduce unscheduled downtime, leading to better productivity and lower production costs. Summarizing, PdM can be viewed as a technique for monitoring the operating condition of a system, providing data that enables the

maximization of the time intervals between repairs and minimizes the unscheduled stoppage time due to machine failures.

Nowadays, PdM is moving from Condition-based Maintenance to Analytics-and IoT-based PdM (Fogoros 2018; Scully 2017). The mains reasons are that modern machinery often enable real-time data capture and at the same time, the cost of implementing embedded systems has been and continues to do so, significantly reduced (Sciban 2017).

In this transition, ML appears naturally as having a significant potential for PdM, and that has been reported in different occasions (Nowitz 2017; Janakiram 2017). ML at its most basic definition can be summarized as a way of applying algorithms to parse data, learn from it and then classify, decide or predict: the machine is trained to find or learn the correlation between the input and the desired output (supervised learning), using large amounts of data captured in the real world. (Irwin 2019) argues that PdM is one of the most relevant areas where ML can be implemented and integrated in the industrial sector: with ML algorithms one can detect asset degradation ahead of time, avoiding breakdowns of resources.

Literature is rich in examples that use ML within the context of PdM. For example, (Heimes 2008) developed a data-driven algorithm to predict the remaining useful life (RUL) of a complex and of nature unknown system as it degrades from an unknown initial state to failure. They applied a Recurrent Neural Network (RNN), trained with backpropagation through time gradient calculations, an extended Kalman Filter training method, and evolutionary algorithms to generate an accurate and compact algorithm.

State of the art Support Vector Machine (SVM) technique was used by (Patil et al. 2015) to estimate RUL of lithium-ion cells. The principal novel contribution is the two-stage (classification– regression) approach for estimating RUL. The proposed approach reduces the input parameter set to a minimal set of critical features, and enables the regression to be much accurate, in addition to reducing the overall simulation time: the addition of a classification step before the regression step eliminates the need to perform regression across the complete battery life cycle data. Hence due to introduction of this step, heavy computations can be eliminated.

(Wang and Mamo 2018) proposed a hybrid model to improve the accuracy of RUL prediction: SVM is used for prediction of battery state-of-health (SOH) based on capacity fade data collected during accelerated ageing (charge/discharge cycling) of batteries and the Differential Evolution (DE) heuristic is used for obtaining the SVM kernel parameters.

(Li, Ding, and Sun 2018) proposed a deep learning method based on Convolution Neural Networks (CNN) and dropout technique to carry out the task of estimate the RUL of aero-engine units accurately. Their model was able to well predict the RUL in the lifetime of the engine units, especially for the late period close to failure.

The objective of this work is to investigate the use of ML in the context of PdM. The main goal is to answer the question about how one can build a ML model to start with initial testing, focusing in using supervised learning for regression problems regarding event detection and prediction, and more precisely, in transportation systems.

Most modern transportation systems rely on sophisticated management software to improve quality of service. TimeKeeper by EFACEC (TK) is such a system: it helps operators to run buses and light rail vehicles on time and to optimize network operation. TK implements the following services:

- Fleet and driver operational management: manages fleet and driver assignment, monitors in real-time all information needed to guarantee an efficient use of resources and to provide batch data and statistic on performed transport-services.
- Real-time service optimization: TK actively monitors the quality of service delivered to passengers. The dispatchers in the control room, can improve the quality of service applying regulation maneuvers, i.e., designing detours, adding, or modifying timetables or transport-services. The passengers are informed of such changes in real-time.
- Environment: TK is fully compatible with different signaling and traffic systems and can trigger switch-point commands and/or priority request based on vehicle location and current timetable, minimizing delays (increasing commercial speed), and saving fuel or energy.

The reports and messages captured or generated within such a system are often stored in data warehouses and can be used to perform automated data mining. TK events can be described as a set of attributes: type of event, level, value, timestamp, etc. These different classes of past events can be used to detect and predict future occurrences of the same class of events.

This is a hard problem, as this is a very complex system, made from the integration of a myriad of different equipment and systems that work together in a network, and events can occur at different points of this network. Moreover, large databases of events are relatively new, so the problem is relatively recent, and to the best of our knowledge there is little published work regarding the use of ML for PdM when related to transportation systems. In this article, a ML data-driven approach, based in deep learning techniques is presented to estimate future events within the TK system. ML data-driven approaches are chosen since they make use of existing and past observations (historical time-series data) for estimating future values. The data-driven approaches are much faster and easier to implement compared to model-based approaches, even though they require a large number of observations in the training phase and the prediction model is non-transparent (Razavi-Far et al. 2018). In this context, artificial neural networks (ANN) can be used to achieve data-driven models and require only easily obtainable values. ANN performs despite multi-dependences and affirms the easy adaptivity of the method to other problems. As any learning mechanism, ANN requires many diverse data to be effective.

This report is organized as follows; in section 2.1, the theory behind LSTM recurrent neural networks is given and in section 2.1.1, a model architecture based in stacked LSTMs is proposed and detailed. In section 2.2, a framework that can be used to build ML models is discussed: the feature selection, the training process and the hyperparameter configuration are all explained in detail. A software platform is also reported and described. In section 3, the experimental results are presented and discussed and in section 4, the final conclusions are reported.

2. Formulation

In this section, the proposed deep learning model design (and implementation) used to predict multiple classes of TK events, is presented.

The available datasets represent TK events, which can be described as a set of attributes: type of event, level, value, timestamp, etc. The main goal is from these different classes of past events, detect and predict future occurrences of the same class of events. To achieve this goal, TK event data is represented in a way that divides time into intervals and aggregates the various event attributes over those intervals, creating a time-series (please go to section 2.2.1, for a more detailed description). Time-series are fixed rate samples, so they consider fairly both bursts of activity and intervals with very low activity intervals: a time-series will therefore be a sequence of vectors, where each vector can be divided into the variables to be learned (the *y* variables) and the variables that are just used as data sources (the *x* variables). Finally, the main goal is to learn or to predict that a type of event occurred in a time slot. Formally, the approach used in this work is to learn $y[k] = f(x_1[k], \dots, x_n[k])$, which reduces to standard supervised learning, and deep learning techniques (i.e., Long-Short Term Memory (LSTM) neural networks) are used to learn the transfer function $f(.)$.

2.1. Long Short-Term Memory Networks

LSTMs are a kind of 'special' RNNs, capable of learning long term dependencies. They were introduced by (Hochreiter and Schmidhuber 1997) and work tremendously well on a large variety of problems, and are now widely used.

Like all recurrent neural networks, LSTMs have the form of a chain of repeating modules of neural network. But, although LSTMs have this chain like structure, the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way (Figure 1) (Olah 2015).

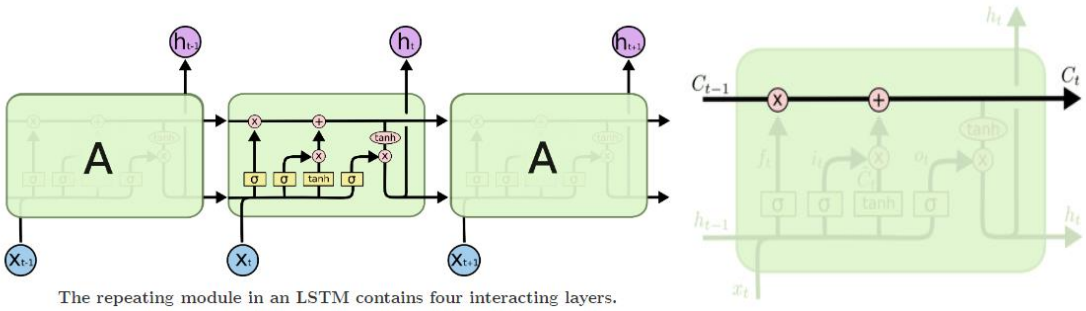

Figure 1: taken from Colah's Blog (Olah 2015)

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates (Figure 1) (Olah 2015).

As shown at the right in Figure 1, C_t is the cell state at time t and C_{t-1} is the cell state at time t-1, σ is the gate.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through", while a value of one means "let everything through!".

An LSTM has three of these gates (forget gate, input gate and output gate), to protect and control the cell state (Olah 2015). Forget gates exist in LSTMs exactly to deal and avoid the long-term dependency problem. Since the gates can prevent the rest of the network from modifying the contents of the memory cells for multiple time steps, LSTM networks preserve signals and propagate errors for much longer than ordinary RNNs. By independently reading, writing, and erasing content from the memory cells, the gates can also learn to attend to specific parts of the input signals and ignore other parts. These properties allow LSTM networks to process data with complex and separated interdependencies and to excel in a range of sequence learning domains. A formal definition of the inner working of LSTMs is presented next.

Let's consider an RNN that is based in a recurrent connection where the hidden state at time t, h_t, is a function of the hidden state at time t-1, h_{t-1} and the input value at time t, X_t:

$$h_t = \sigma(W[h_{t-1}, X_t] + b) \quad (1)$$

The LSTM adds a cell state C_t in addition to the hidden state, h_t. The computation is then distributed into several gates executed sequentially: first, the forget gate determines the pieces of the long-term memory to continue remembering and the pieces to ignore using the new input:

$$f_t = \sigma\big(W_f[h_{t-1}, X_t] + b_f\big) \quad (2)$$

Next, the input gate determines the information that should be extracted from the current input data, X_t:

$$\xi_t = tanh\big(W_\xi[h_{t-1}, X_t] + b_\xi\big) \quad (3)$$

$$y_t = \sigma\big(W_y[h_{t-1}, X_t] + b_y\big) \quad (4)$$

Finally, the output gate updates the cell state and the hidden state, using the past and present information:

$$C_t = f_t C_{t-1} + y_t \xi_t \quad (5)$$

$$o_t = \sigma(W_o[h_{t-1}, X_t] + b_o) \quad (6)$$

$$h_t = o_t tanh(C_t) \quad (7)$$

Where: W_f, W_ξ, W_y and W_o are weight matrices; b_f, b_ξ, b_y and b_o are bias matrices; σ is the sigmoid function.

2.1.1. Model architecture

Different model architectures were investigated and are resumed on Table 1.

In Table 1, the 1st column represents the model's name, and the 2nd column gives a description about the different layers, and types of layers used in the corresponding model.

Table 1: implemented LSTM-based models

Model name	Description
LSTMx1x32	1x LSTM layer with 32 neurons, followed by 1x dense layer with single neuron output
LSTMx1x64	1x LSTM layer with 64 neurons, followed by 1x dense layer with single neuron output
LSTMx1x256	1x LSTM layer with 256 neurons, followed by 1x dense layer with single neuron output
LSTMx1x32 + 2xDNN	1x LSTM layer with 32 neurons, followed by 2x dense layers with single neuron output
LSTMx2x8	2x LSTM layers, both with 8 neurons, and followed by 1x dense layer with single neuron output

These architectures are represented in Figure 2.

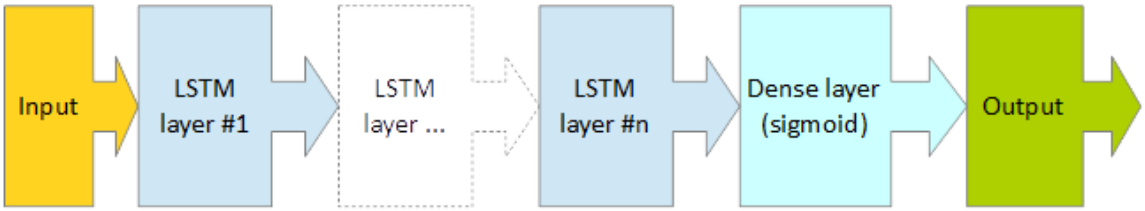

Figure 2: stacked LSTM architecture

The models are implemented as a stacked LSTM architecture. The final layers are dense neural network layers with 1 output neuron. For the dense layers, the chosen activation function is a sigmoid function.

Given that LSTMs operate on sequences of data, the addition (stack) of more layers adds levels of abstraction of input observations over time. In effect, chunking observations over time or (potentially) representing the problem in different time scales (Graves, Mohamed, and Hinton 2013). Stacked LSTMs where first introduced by (Graves, Mohamed, and Hinton 2013): the authors found that the depth of the network was more important than the number of memory cells in a given layer.

Stacked LSTMs are now a stable technique that can be used to solve sequence prediction problems (like the one in this article!). A stacked LSTM architecture can be described as an LSTM comprised of multiple LSTM layers. Each layer provides a sequence output (instead of a single value) to the next LSTM layer. Specifically, one output per input time step.

In this article, the input sequence (X) is represented by a configurable number of time steps of the tuples $X_i^j = \{count_i, min_i, max_i, average_i\}$. The output is represented by the next configurable number of time slots of the output time-series. For better understanding, consider that for an input sequence of length 50, represented by $\{ X_i^j, X_{i+1}^j, \dots, X_{i+49}^j \}$, where X_i^j is the input data for the i th time slot, the output will be a sequence represented by

$\{ Y^j_{i+50}, Y^j_{i+51}, \ldots, Y^j_{i+59} \}$, representing the output values for the next 10 time slots. Basically, this LSTM model will be able to predict the next occurrence of a TK event calls within a window of 10 time slots, in the future.

2.2. Applied framework

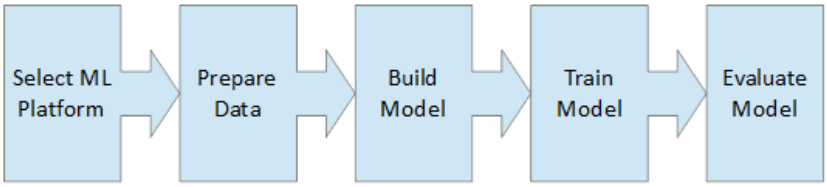

Figure 3: a framework for building ML models

In Figure 3, the framework used to carry out the implementation of the proposed model architecture of section 2.1.1 is represented.

When building an ML model, the first aim is to get the model to fit the training datasets. Once enough results are available, the model can be tested on other independent data. To achieve this, a primary decision is to select the best available platform (programming language, development tools, software libraries, ML engines, etc.). In this report, Python programming language is the chosen tool to achieve the required results. (Luis Pedro Coelho 2018; Vahid Mirjalili 2017) present ML and Python as a dream team. Within the Python frameworks for ML, Keras (running on top of TensorFlow) is selected, since it provides a way for fast experimentation with easy and fast prototyping.

During data preparation, the steps described in section 2.2.1, were applied, converting the original dataset into a set of time-series, per event class. The original dataset was divided into 3 smaller datasets used for training, validation, and testing of the model, representing 60%, 20% and 20% of the original dataset. Basic standardization of data was enforced, scaling features to lie between a given minimum and maximum value, in this case between zero and one, or so that the maximum absolute value of each feature is scaled to unit size. Finally, the input/output sequences described in section 2.1.1 are created in the correct format, usable by the Keras model.

The next step is to create and build the proposed deep learning model. The selected Keras model type is the Sequential one: a model where one stack the layers, or in other words, add one and one layer, which serves well the architecture in section 2.1.1. Further, after each LSTM layer, dropout was used to avoid overfitting. ANNs trained on relatively small datasets can overfit the training data. This has the effect of the model learning the statistical noise in the training data, which results in poor performance when the model is evaluated on new data, i.e., a test dataset. Generalization error increases due to overfitting. Dropout regularization was first introduced by (Hinton et al. 2012) and is a computationally cheap way to regularize a deep neural network. Later work by (Srivastava et al. 2014) clearly demonstrated its improved performance when training ANNs: dropout works by probabilistically removing, or dropping out, inputs to a layer, which may be input variables in the data sample or activations from a previous layer. It has the effect of simulating many networks with very different network structure and, in turn, making nodes in the network generally more robust to the inputs. The loss function chosen for training the model was the Binary Cross-Entropy function, as we are trying to optimize binary data.

For the optimization algorithm, Adam optimizer is the best choice when training RNNs, since it is currently the stochastic gradient optimizer most used in deep neural networks (DNN), with descent at default parameters. The learning rate during training was set to 0.001. In Table 2, a summary of the used training hyperparameter's configuration is presented.

Table 2: training configuration

Hyperparameter	Value
Optimization algorithm	Adam Optimizer
Learning rate	0.001
Dropout	0.2
Number of epochs	100
Batch size	128

The 1st column in Table 2 represents the hyperparameter name or description and in the 2nd column, the configured value used during the training phase of the model, is given.

Finally, during the evaluation phase, the quality of the final model is computed both numerically and by visual inspection. The numeric criteria used was the Area Under the Curve for the Receiver Operating Curve (AUC-ROC) metric. This metric relates the false-positive and true-positive rate. It works well for balanced data. A typical rule of thumb is that an AUC around 0.5 represents model randomness, 0.6 AUC significantly better than random model and over 0.7, a good classifier.

2.2.1. Feature extraction

The available datasets represent different classes of TK events, which can be described as a set of attributes: type of event, level, value, timestamp, etc. TK event data is then represented in a way that divides time into intervals and aggregates the various event attributes over those intervals, creating a time-series. For example, Table 3 shows the grouping of TK events over a time interval and Figure 4 orders and translates the events to a bar plot.

Table 3: Different TK events: acronym, event description, number of occurrences

TK event acronym	Description	Number of occurrences
CABIN	Active driver cabin failure	27
MPU	Communication failure with the MPU	533
CONSOLE	Communication failure with driver's console	219
CAPSYS	Communication failure with onboard signaling antenna	210
AVL	Communication failure with operator workstations	146
VEHICLE	Data communication failure with onboard computer	1231
PCE	Failure of an onboard passenger communication module	153
ODOMETER	Odometer failure	45
PABS	Public announcement system failure	25
WIFI	Wifi radio failure	1004

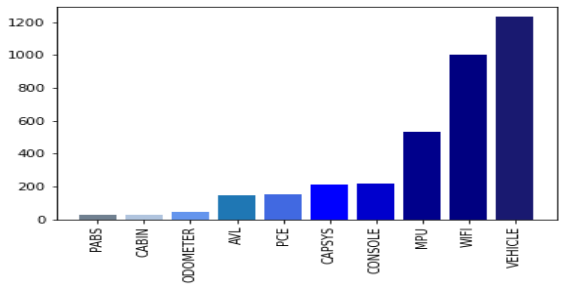

Figure 4: TK events bar plot over a time interval

This bar plot is interesting and gives a better view of the occurrences of different events in the dataset. It also gives an indication of the events that will be the hardest to forecast. For example, PABS and CABIN are the rarest events in the dataset, which could make them harder to predict.

Time-series are sequences of samples taken at a constant pace. Since the original dataset is composed of multiple classes of events with a timestamp attribute, the conversion to time-series requires the grouping of events by time unit or time slot. The used algorithm is as follows:

- From the original dataset, define a time window [t_0, t_1] for analysis, where t_0, is the initial timestamp and t_1, the final timestamp.
- Within this interval, convert every event timestamp as the number of seconds starting from t_o. So, $t_0 = 0$ and t_1 = some random number of seconds.
- Group the events by time slot, where the time slot can be defined as an hour or a day. Note that, after this grouping, each time slot is represented by a set of different classes of TK events, where each class has some number of events in total.
- Next, for every time slot, some statistics (*count*, *min*, *max*, *average*) are taken. This new set of statistics (the *x* variables), for each class of TK event and per time slot, represent the sample *i* (i.e., $X_i^j = \{count_i, min_i, max_i, average_i\}$) of the input sequence representing the input time-series. Note that now, a time-series of different samples *i*, per event class *j*, exists.
- Finally, the output time-series is represented as boolean values (the *y* variables, i.e., the values to be learned). For each class of event and each time slot, one detects if that class has any event present on that time slot, and this information is represented as TRUE (value 1), or not, and this information is represented as FALSE (value 0).

3. Experimental results and discussion

In this section, the results obtained for each model used to predict multiple classes of TK events, is presented.

During the training phase, different algorithms (models) were experimented to help obtain the best results. The results shown in Table 4 are based on the AUC metric, as stated above.

Table 4: AUC results for different LSTM models

TK event acronym	LSTMx1x32	LSTMx1x64	LSTMx1x256	LSTMx1x32 + 2xDNN	LSTMx2x8
CABIN	None	None	None	None	None
MPU	**0,9129**	0,6806	0,8826	0,6629	0,4400
CONSOLE	0,8241	0,9338	0,9681	0,5418	**0,9811**
CAPSYS	0,9842	0,7126	**0,9897**	0,8404	0,8540
AVL	**0,8112**	0,6720	0,5594	0,3399	0,7938
VEHICLE	0,7060	**0,9152**	0,6472	0,4095	0,8567
PCE	0,4821	0,8127	**0,9839**	0,9319	0,4703
ODOMETER	0,4157	0,0105	**0,9777**	0,9409	0,5791
PABS	None	None	None	None	None
WIFI	0,8597	0,8736	0,9582	0,8604	**0,9927**

In Table 4, the 1st column represents the TK event class acronym, and the following columns give the AUC metric values obtained for each model used, on a particular class of events. A "None" value means that the AUC metric was not possible to compute. The bold values represent the best model, for each TK event class.

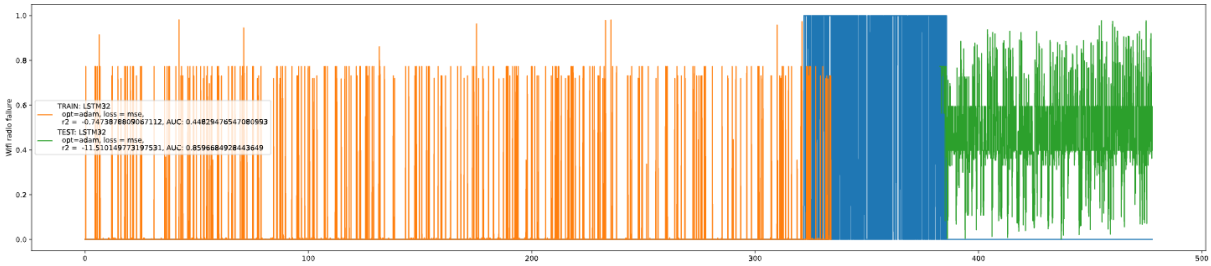

Figure 5: Result for LSTMx1x32 for Wifi radio failure

In this example (Figure 5), blue corresponds to real values in the dataset, orange corresponds to training and green corresponds to forecasted events.

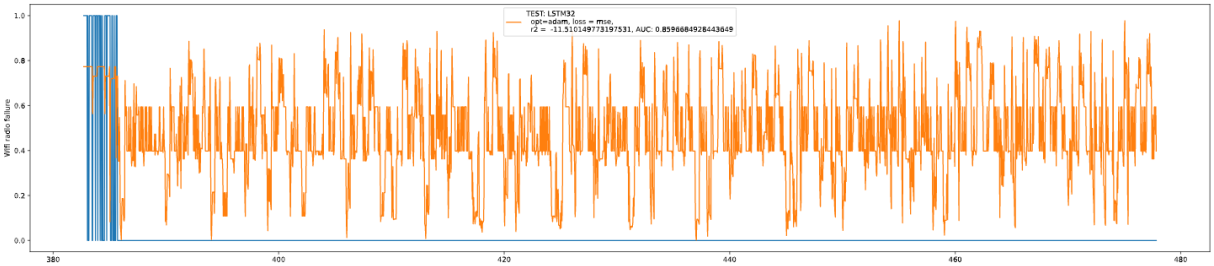

Figure 6: Results for LSTM1x32 for Wifi radio failure (test only)

Figure 6 shows another plot, this time only for the test section. This is the section that is important for predictive maintenance, where future events will occur and having only the tests part of the model gives us a better view of it. Each value in orange in the plot represents a probability value of the event to happen.

The models get very good results overall, except for 2 events: CABIN and PABS. The bad results for these two models were expected and can be explained since these are the event classes that occur the least in the TK dataset. This difference of occurrences could be seen in Figure 4 of section 2.2.1.

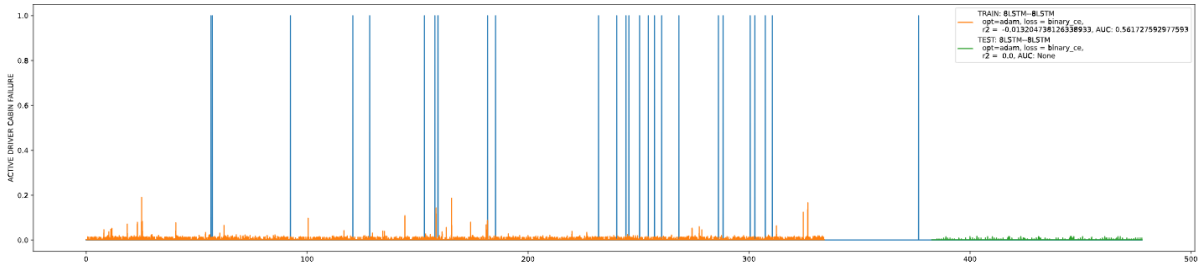

Figure 7: Example of a "None" value for AUC

Figure 7 shows an example of a "None" value presented on Table 4. As stated before, the AUC metric relates the false-positive and true-positive rate. With no real events (blue) in the test data, there is no way to compute the AUC value.

4. Conclusions

In this paper, the use of ML in the context of PdM, was investigated. The main goal was to answer the question about how one can build a ML model to start with initial testing, focusing in using supervised learning for regression problems regarding event detection and prediction, and more precisely, in transportation systems.

To achieve this, a dataset of different event classes was built, using the information gathered within the TimeKeeper by EFACEC system. The TimeKeeper by EFACEC is a software tool that helps operators to run buses and light rail vehicles on time and to optimize network operation. This dataset is then transformed into a time-series, to be used in the training of different LSTM model's, used to predict future events.

When using time-series, the problem is reduced to whether an event occurred at a timeslot. Then, the models are implemented as stacked LSTMs for learning. The values obtained by the majority AUC values for each model are very good.

One rule that can be applied is that the more data we have, the better the results are expected to be. This can be applied precisely to CABIN and PABS events, being the rarest events in the dataset and the ones that no model was able to forecast, since no pattern was captured. With more available data from CABIN and PABS, the results for those alarms could be improved.

The experiments also revealed that building LSTM models with more layers, do not always produce better predictions, while are harder to train (more computation power and bigger training times). A single layer LSTM model, with enough nodes, usually results in accurate predictions.

Overall, the average AUC of the different LSTM models was 0.9456, which is an exceptionally high value, proving the validity of the approach reported in this paper.

5. References

1. Amruthnath, N., and T. Gupta. 2018. "A research study on unsupervised machine learning algorithms for early fault detection in predictive maintenance." 2018 5th International Conference on Industrial Engineering and Applications, ICIEA 2018.
2. Chiu, Y. C., F. T. Cheng, and H. C. Huang. 2017. "Developing a factory-wide intelligent predictive maintenance system based on Industry 4.0." *Journal of the Chinese Institute of Engineers, Transactions of the Chinese Institute of Engineers,Series A/Chung-kuo Kung Ch'eng Hsuch K'an* 40 (7): 562-571. https://doi.org/10.1080/02533839.2017.1362357. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85028555499&doi=10.1080%2f02533839.2017.1362357&partnerID=40&md5=fc1e6210f04ddc98593323e1de267a4f.
3. Fogoros, Lucian. 2018. "Connected sensors & machine learning - two current trends in predictive maintenance." http://iiot-world.com/predictive-maintenance/connected-sensors-ml-two-current-trends-in-predictive-maintenance/.
4. Graves, Alex, Abdel-rahman Mohamed, and Geoffrey E. Hinton. 2013. "Speech Recognition with Deep Recurrent Neural Networks." *CoRR* abs/1303.5778. http://arxiv.org/abs/1303.5778.
5. Heimes, F. O. 2008. "Recurrent neural networks for remaining useful life estimation." 2008 International Conference on Prognostics and Health Management, PHM 2008.
6. Hinton, Geoffrey E., Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2012. "Improving neural networks by preventing co-adaptation of feature detectors." *CoRR* abs/1207.0580. http://arxiv.org/abs/1207.0580.
7. Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. "Long Short-Term Memory." *Neural Comput.* 9 (8): 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735.
8. Irwin, Richard. 2019. "Revolutionizing Reliability." https://reliabilityweb.com/articles/entry/predictive-maintenance-and-machine-learning-revolutionizing-reliability.
9. Janakiram, MSV. 2017. "How Machine Learning Enhances The Value Of Industrial Internet of Things." https://www.forbes.com/sites/janakirammsv/2017/08/27/how-machine-learning-enhances-the-value-of-industrial-internet-of-things/#73c94163f389.
10. Li, X., Q. Ding, and J. Q. Sun. 2018. "Remaining useful life estimation in prognostics using deep convolution neural networks." *Reliability Engineering and System Safety* 172: 1-11.

https://doi.org/10.1016/j.ress.2017.11.021. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85036657786&doi=10.1016%2fj.ress.2017.11.021&partnerID=40&md5=6e1681893a13f3e5829eb0380e73f83d.

11. Luis Pedro Coelho, Willi Richert, Matthieu Brucher. 2018. *Building Machine Learning Systems with Python*. 3rd ed.: Packt Publishing.
12. Nowitz, Avi. 2017. "The Economics of the Smart Factory: How does Machine Learning Lower the Cost of Asset Maintenance Part 1." https://www.presenso.com/single-post/2017/05/24/the-economics-of-the-smart-factory-how-does-machine-learning-lower-the-cost-of-asset-maintenance-part-1/.
13. Olah, Christopher. 2015. "Understanding LSTM Networks." http://colah.github.io/posts/2015-08-Understanding-LSTMs/.
14. Patil, M. A., P. Tagade, K. S. Hariharan, S. M. Kolake, T. Song, T. Yeo, and S. Doo. 2015. "A novel multistage Support Vector Machine based approach for Li ion battery remaining useful life estimation." *Applied Energy* 159: 285-297. https://doi.org/10.1016/j.apenergy.2015.08.119. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84941893545&doi=10.1016%2fj.apenergy.2015.08.119&partnerID=40&md5=4b60598acbb4531ef3cf332a8ce55c01.
15. Razavi-Far, R., S. Chakrabarti, M. Saif, E. Zio, and V. Palade. 2018. "Extreme Learning Machine Based Prognostics of Battery Life." *International Journal on Artificial Intelligence Tools* 27 (8). https://doi.org/10.1142/S0218213018500367. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85058948338&doi=10.1142%2fS0218213018500367&partnerID=40&md5=147e50e5bb6e74222ba5a0c152a176f4.
16. Sciban, Rowan. 2017. " 6 Tools for a Successful Predictive Maintenance Program." https://us.hitachi-solutions.com/blog/6-tools-for-a-successful-predictive-maintenance-program/.
17. Scully, Padraig. 2017. "New Report Indicates US$11 Billion Predictive Maintenance Market By 2022, Driven By IoT Technology And New Services." https://iot-analytics.com/report-us11-billion-predictive-maintenance-market-by-2022/.
18. Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. 2014. "Dropout: A simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research* 15: 1929-1958. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84904163933&partnerID=40&md5=b865fd654b3befc5d829dbe5d42b80c3.
19. Vahid Mirjalili, Sebastian Raschka. 2017. *Python Machine Learning*. 2 ed.: Packt Publishing.
20. Wang, Fu-Kwun, and Tadele Mamo. 2018. "A hybrid model based on support vector regression and differential evolution for remaining useful lifetime prediction of lithium-ion batteries." *Journal of Power Sources* 401: 49-54. https://doi.org/https://doi.org/10.1016/j.jpowsour.2018.08.073. http://www.sciencedirect.com/science/article/pii/S0378775318309418.