Intelligent time-series forecasting and event prediction for Predictive Maintenance in IT systems

Pedro Moreira
Instituto Superior Técnico
pedro.santiago@tecnico.ulisboa.pt

***Abstract*—Nowadays, most Information Technology systems already perform Condition-based Maintenance, which provides an overview of a system's condition in real-time and stores its behaviour (in the form of time-series). A Predictive Maintenance approach can use this historical data to apply planning corrective maintenance based on predictions about a system's evolution. This work intends to provide useful research in the scope of Predictive Maintenance through the study of the best approaches and algorithms to perform time-series forecasting and event prediction within the Information Technology domain. The state-of-the-art intelligent methods for time-series modelling were studied, as well as the most promising methods developed for other domains with existent literature ahead of time-series (like Natural language Processing). The main focus was on Machine Learning techniques - from the primary Feed-forward Neural Networks all the way to the more recent and complex Transformers. For the time-series forecasting, none of the experimented models performed satisfactorily. However, it was notable that with the increase of complexity and size of the model's architectures, they learned to output "dummy" forecasts like a naive approach or a constant value, which although not useful, minimise the evaluation metrics. For the event prediction problems, a preprocessing step to detect oscillations in the input datasets significantly boosted the algorithms' performances. Furthermore, the results obtained were not ideal but satisfactory enough to be useful, and the model that showed the best results was the Feed-forward Neural Network. Finally, it is possible to adjust the predictions' sensitivity with the tuning of a data preprocessing factor.**

***Index Terms*—Predictive Maintenance; Information Technology systems; Time-series forecasting; Event prediction; Computational Intelligence.**

I. Introduction

Nowadays, most Information Technology (IT) systems are able to perform Condition-based Maintenance (CBM), which means that they are capable of monitoring the condition of their components in real-time and decide what maintenance is needed. The downside of CBM is that it only orders maintenance actions when certain indicators show signs of decreasing performance or upcoming failure, which means that problems might have already occurred – a problematic system is not reliable, and lack of reliability is not a good sign for enterprise profitability. On the other hand, since IT systems implement CBM, they are already capable of keeping track of their functioning and store their behaviour history. This way, a helpful monitoring approach can make use of this information to predict possible failures in time to avoid/soften them and predict components' future behaviours to, timely, take appropriate precautions. This approach is known as Predictive Maintenance (PdM), which refers to planning corrective maintenance based on predictions about the evolution of a system.

The design of PdM techniques aims to determine the condition of a system and its components ahead of time. By looking forward and knowing what failures are likely to occur, it is possible to schedule adjustments and repairs to apply them before assets fail and/or the system evolves to an unwanted state – these preventive actions will provide a stable environment and an increased assets life. This way, a useful PdM approach shall be capable of providing the means to improve productivity, product quality, and overall effectiveness. Looking from an industry point of view, these improvements achieved by PdM will lead to a vast range of benefits [1] that can both save money and maximize efficiency, such as:

- Reduction (if accurate enough, near elimination) of unscheduled equipment downtime caused by equipment or system failure;
- Better asset management that results in an increased production capacity and labour utilisation;
- Increased equipment lifespan and more economical use of maintenance workers that significantly reduces maintenance costs.

A private company's monitoring software tool for IT systems and components integrated the PdM algorithms developed in this thesis. The private company is - Identity - and this thesis engages in a partnership with them. The monitoring platform performs CBM on metrics such as network utilisation, Central Processing Unit (CPU) load, disk space consumption, etc. As most of IT monitoring platforms, the collected data on these metrics is stored chronologically. Identity made available for this study historical data on hundreds of metrics dating back to more than five years. This data came from a very distinct set of machines and users – be it for internal production inside the company or for external clients/companies that work in different industries and manage systems from very diverse environments.

Since the historical data kept by monitoring platforms from IT systems is often stored chronologically and with a timestamp attached to every collected metric, this data can be looked at and analysed as time-series. Time-series analysis is one of the areas with the biggest potential in PdM, and historical data, which is its feedstock, can be acquired very easily by IT systems – most of them already perform this data collection with the implementation of CBM. The use of Artificial Intelligence (AI) and other intelligent methods in PdM has been growing in the past few years, but the majority of the companies did not adopt it yet [2]. This way, the research on intelligent methods to analyse time-series in a PdM perspective can be not only a very recent and interesting development, but it can also have a significant practical impact in the real-world environments.

There are several different approaches that one can take towards the development of a PdM solution. After a PdM background study applied to modern IT systems needs, and a strategic fit with the Identity's monitoring software, this dissertation will tackle two separate problems in the PdM world that were found to be the most convenient and advantageous to be integrated with the monitoring of IT systems:

- **Time-series forecasting** – this solution will consist in the development of methods to predict the future values of an IT system's metric, based on its past values and the past values of multiple other metrics (the system's history), for example, predicting the CPU load for tomorrow at 2 pm.
- **Event prediction** – this solution will consist in the development of methods to predict the occurrence of sporadic events before they actually occur, based on the event's previous occurrences, some others events previous occurrences and the history of some system's metrics, for example, issuing a warning four hours before the system collapses due to lack of free memory.

II. Related Work

In the literature study of this work, the research after the state-of-the-art intelligent methods for modelling time-series was led. Beyond that, the most promising models developed in other domains, such as Natural language processing (NLP), and with (yet) scarce research for time-series, were also adapted and used this work with time-series datasets.

The models tested were the following: (a) Feed-forward Neural Network (FNN); (b) Recurrent Neural Network (RNN) - introduced by J. L. Elman [3]; (c) Long Short Term Memory (LSTM) - introduced by S. Hochreiter and J. Schmidhuber [4]; (d) Gated Recurrent Unit (GRU) - introduced by K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio [5]; (e) encoder-decoder variants - introduced by I. Sutskever, O. Vinyals, and Q. V. Le [6]; (f) encoder-decoder with the attention mechanism variants - introduced by D. Bahdanau, K. Cho, and Y. Bengio [7]; (g) the Transformer - introduced by A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin [8]. The variants are referred to the three possible cells: RNN, LSTM, and GRU. For the time-series forecasting problems, the Auto-regressive Integrated Moving Average (ARIMA) method was also tested and compared, as it is the most common benchmark in this field.

III. Dataset Description

A. Time-series Forecasting

A monitoring platform collected the datasets available for this dissertation from "up and running" real-world IT systems. The platform defines each metric collection as an item, and an item is mainly (i.e. relevant for this dissertation) characterized by two settings: (a) the actual metric, for example, the CPU load of computer X; (b) the collection period, for example, five minutes (5min). Every collection is then performed, for each period, only at the instant that the period clocks. For the given example, if one CPU load value is collected at 14h 00min, the next value is going to be the CPU load at the instant 14h 05min, and not the average load of the CPU between 14h 00min and 14h 05min.

The monitoring platform has two ways of storing the data regarding the items collected – history and trend – each kept separately in the database:

- **History** – keeps each collected value for a pre-defined period of time (for example, data older than one month will be discarded by a housekeeper). An example of a portion of an item's history can be seen in Table I (with a period of 1min).

TABLE I
HISTORY TABLE EXAMPLE OF CPU LOAD [%].

Time	CPU load [%]
15h 03min 00s	15.32
15h 04min 00s	18.01
15h 05min 00s	15.44

- **Trends** – basically a historical data reduction mechanism which stores minimum, maximum, average and the total number of values per every hour for numeric data types. An example of a portion of an item's history can be seen in Table II.

TABLE II
TREND TABLE EXAMPLE OF CPU LOAD [%].

Time	Min	Max	Avg	Values count
11h 00min 00s	15.32	63.35	21.33	20
12h 00min 00s	18.01	71.27	25.24	20
13h 00min 00s	15.44	54.20	20.05	20

After an agreement with Identity on the most useful approach, the methods studied and fitted in this work were set to forecast the average value of trends for time horizons of two hours and beyond. This way, the input data used to compute the forecasts is prevenient from trends storage, not history.

1) Event Prediction: Similarly to the time series datasets, every event occurrence will come with a timestamp attached stating when the event occurred. However, there are only values when an event occurs, which is supposedly random and without any respect for periods. In order to have these datasets interpreted and analysed as time-series, they need some data preprocessing – those procedures will be explained later in Section IV-B1. Events datasets will be in the form of Table III where, when a problem is triggered (event occurrence), a the platform stores a value of 1 with its correspondent timestamp; when the problem is solved (or stopped existing), it stores a value of 0 with its correspondent timestamp as well.

TABLE III
EVENT DATASET EXAMPLE.

Timestamp	Event
2020-01-05 01:00:03	1
2020-01-05 01:07:29	0
2020-01-08 04:10:12	1

The available data from the monitoring platform for the event prediction problems are of two types: trends and events. The trend values are of the same type described in Section III-A. The event data is directly related to events, and it comes in the form described above.

The events monitored in this work were related to problems in IT systems, in particular, services shutting down unexpectedly or becoming unavailable. This way, when the system shuts down or becomes unavailable – the event/problem is triggered – and at that instant, a row in the respective event is appended with a value of 1. When the system recovers from the problem, another row is appended with a value of 0.

The available data from trends is related to metrics collected in the same machine/IT system where the events occur.

IV. IMPLEMENTATIONS

A. Time-series Forecasting

1) Data normalization: The most common two approaches for normalizing input datasets for Machine Learning (ML) algorithms are the standardization and the and the Min-Max scaling. There is no clear winner when choosing the data normalization method (the normalization itself does not even improve the performance that much). However, both standardization and Min-Max scaling were tested in all cases but showed similar results. The results presented in Section V were obtained with the Min-Max scaling normalization.

2) Sliding Window: The most common approach to transform a sequential time-series dataset into these input-output pairs, which is the same one used in this work, is the sliding window. This method consists in fixating a window of length $(N_{in} + N_{out})$ in the beginning of the dataset and slide it all the way until the end. Each slide constitutes one trading pair to learn from (one input sequence and one output sequence). A schematic representation of the sliding window approach is exemplified in Figure 1.

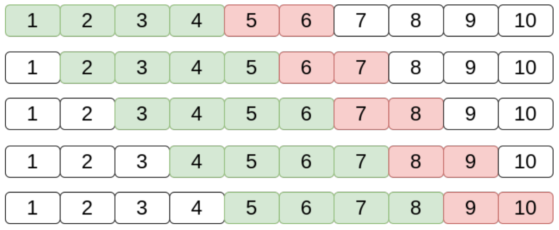


Fig. 1. Sliding window approach.

In this example, the input sequence of length $N_{in} = 4$ is represented in green and the output sequence of length $N_{out} = 2$ is represented in red. Since the monitoring platform where the algorithms were implemented would become more versatile with variable (user-defined) forecasting horizons, a range of lengths within $[2h, 24h]$ was tested in this work for the output sequences.

After testing all of the methods with different sequence lengths (input and output), the results showed that input sequences longer than $24h$ did not improve any of the model's performances and that for output sequences of $6h$, was length sufficient to illustrate how the models behave in multistep-ahead forecasts. For output sequences until $6h$, the forecasts did not lower the accuracy significantly. As such, the results presented in Section V for time-series forecasting were conducted with $N_{in} = 24; N_{out} = 6$.

3) Hyperparameters of the intelligent method: The hyperparameters optimised in the ML models applied in this are the following: (a) learning rate, (b) loss function, (c) optimisation solver, (d) activation functions, (e) number of hidden nodes and (f) number of stacked layers. For the ARIMA model, the hyperparameters to be predetermined are the constants p, q and d – $ARIMA(p, q, d)$.

In this work, the optimal hyperparameters were manually deduced trial-and-error.

The most common optimisation solvers are: (a) the *sgdm* [9], (b) the *rmsprop* [10] and (c) *adam* [11]. All these three optimisation solvers were thoroughly tested across all the ML models, and the *adam* showed either equal or superior behaviour in all tests. As such, the results of Section V were obtained with the *adam* optimisation solver. The loss function used in all Artificial Neural Network (ANN) for the time-series forecasting problems was the Mean Square Error (MSE), which seems to be a common choice for this type of predictions. The learning rate used was $lr = 0.05$; tests with other values between $[0.01, 0.1]$ were experimented but did not change the results. For last, the following activation functions - (a) Linear, (b) Rectified Linear unit(ReLu), (c) Sigmoid and (d) Hyperbolic tangent - were tested, one for each layer of each ANN (all combinations), but, except using only linear units (that showed worst results), the performances were similar and thus, this choice is irrelevant.

4) Cross-validation: The cross-validation method of splits the dataset into K segments (folds), equal in size, then $K-1$ folds will be used as the training set for the model to learn from. The remaining fold will be used as the test set. This procedure is repeated, using the same folds split, but always using a different fold for the test set until all the possibilities are carried out. The iteration that showed better results with the underlying test set will be the one from which the final model will be taken. This procedure is schematically represented in Figure 2, with $K = 5$.

Furthermore, given the time ordering present in time-series, the Cross-validation method might ignore the sequential nature of time. Namely, instead of just wanting the model to generalise to unseen new data, for time-series forecasting, it is wanted that the model generalises for future data. This way, the method used to secure that the model is only tested for a

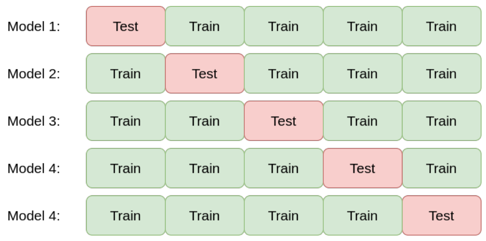


Fig. 2. Cross-validation with $K = 5$ folds.

data latter in time than the training set, is the one represented in Figure 3.

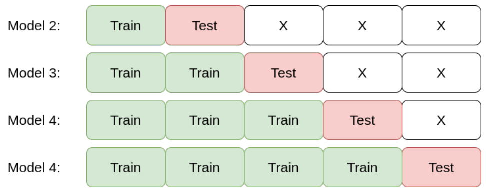


Fig. 3. Cross-validation for time-series with $K = 5$ folds.

5) Evaluation Metrics: The metric chose in this work to evaluate the time-series forecasting models was the MSE, that is mathematically decribed by

$$MSE = \frac{\sum_{i=1}^{N} \left(y_{real}^{i} - y_{forecast}^{i} \right)^2}{N}, \tag{1}$$

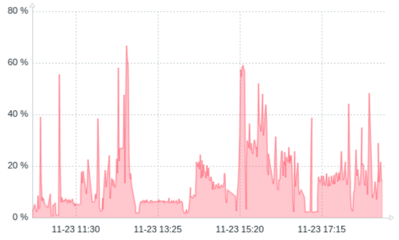
where y_{real} are the true values – the ones present in the test set outputs –, $y_{forecast}$ and the values forecasted by the model, and N is the length of the test set.

6) Period Reduction trial: This experiment was only tested with the ANN architectures.

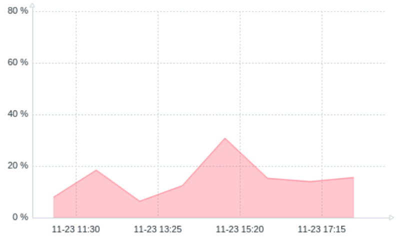
Although the trend tables (one-hour averages) might be able to properly shape data over long horizons, some spontaneous incidents that only happen momentarily could also compose valuable information to compute the forecasts, as they might be related to meaningful events with long term impacts. For example, several CPU spikes during a couple of minutes could indicate the start of several different applications or services, that would increase the average CPU consumption over the next hours. In Figure 4 is a visual example that illustrates how the trends storage misses irregularities in data that are caught by the history storage.

Moreover, given that the metrics collection is not continuous, using the minimum period possible will increase catching spurious occurrences. To test this possibility, items were set to perform collections with the minimum period allowed by the monitoring platform, 1sec. An example of a CPU load collection comparison between a period of 1sec and a period of 1min is present in Figure 5.

It can easily be noted that there are spikes present in Figure 5a, that were not caught with a period collection of one minute – Figure 5b. It must also be noted that this history storage usage is highly incompatible with the monitoring platforms' purpose. According to the platforms' documentation, the history storage is kept as short as possible given that it consumes much more disk space then the trends

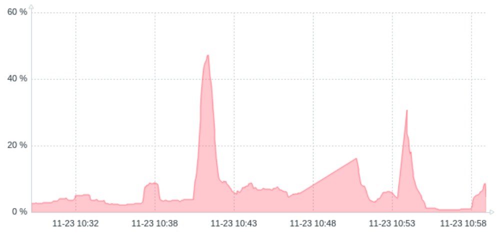


(a) CPU load – history storage.

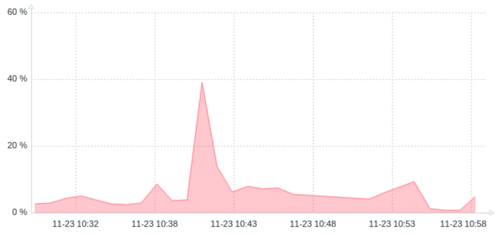


(b) CPU load – trends storage.

Fig. 4. CPU load comparison of the two different storage types: history vs trends; the item's collection period is one minute.



(a) CPU load with a period collection of 1 second.



(b) CPU load with a period collection of 1 minute.

Fig. 5. CPU load comparison of two different collection periods: 1 second vs 1 minute.

storage (which is precisely the point of using trends), and thus keeping history for long periods of time, which would be needed to train the intelligent forecasting models, could be unbearable. Furthermore, the collection of data at very short periods (as one second), weights a significant burden in an IT system, as it needs to be continually polling data from the monitored metrics. A monitoring platform is integrated into an IT system because it can help to maintain it and to keep it reliable. Thus, if the monitoring tools used consume a lot of resources, they will interfere with the system and might weaken it, instead of the opposite. This way, the experiment described in this chapter was conducted to try to understand if eventual forecasts with bad accuracies could be due to the loss of information inherent to the use of trends storage or too large collection periods.

To try to make some meaning out of the spikes noted in datasets like Figure 5a, an approach that aims to count the number of spikes and feed it to the intelligent method was conducted. First of all, given the lack of available history data with short collection periods to train the models, trend tables of 10min averages were manually created to increase the number of time-steps. Using this "artificial" trend table, the same number of time-steps were used for the input and output sequences – $N_{in} = 24$ and $N_{out} = 6$. Then, instead of just using the sliding window approach, described in Section IV-A2, to generate sequences to feed the model, K extra points will be added to the sequence, where K is the number of input features. Each of the K points will be the total count of spikes, $\#spikes$, of the respective feature, during the whole input window time segment. Note that each input is referent to a 10min average, while the spikes count is performed in the whole history data (one value per second). As such, for an input sequence of four time-steps, the input sequence timeline accounts for $4 \times 10 \times 60 = 2400$ seconds, which is the number of points evaluated to count the spikes.

In Figure 6, is an example of the upgraded sliding window just described, but with $N_{in} = 4$ and $N_{out} = 2$.

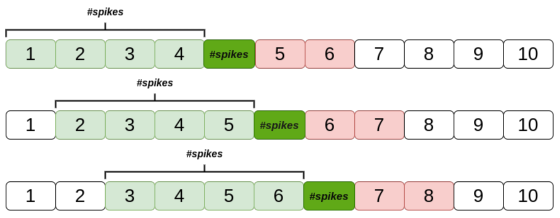


Fig. 6. Sliding window for the period reduction trial.

The spikes count – $\#spikes$ – was calculated with a trial and error approach. Looking at each feature's graphical data, like Figure 5a, define a threshold value, on top of the mean of the whole sequence, above which a spike would be considered abnormal. For example, if the mean of an input sequence is $\mu = 7$ and defined threshold is $T = 30$, for each time that the metric's collected value crosses the value $\mu + T = 37$ during the subject input sequence timeline, the spike count is incremented. For metrics like free memory, where a lower value is the "problematic" and not the other way around (like CPU load), the spike count is incremented when the line $\mu - T$ is crossed. After several trial and error iterations, pick the threshold that led to better results and use it to the final forecasting model.

B. Event Prediction

1) Event to time-series: Given that the event points present in the datasets are not periodical, the intelligent methods studied are not prepared to receive such data as input and compute predictions with it. To convert the original event datasets to periodic sequences suitable for the ANN models, a preprocessing step that results in a binary time-series was applied:

1) Round the timestamps of the events to the closest (past) hour (because the trends have values per hour);
2) Generate a time-series with 0's as values for the same timeline and with the same period (one hour) of the trends metrics that will be used.
3) For the time between every event occurrence and its respective recovery, flip the values of the time-series to values of 1.

After applying this technique, the original events from Table III would be transformed into the time-series table in Table IV.

TABLE IV
TIME-SERIES EVENT DATASET EXAMPLE.

Timestamp	Event
2020-01-05 00:00:00	0
2020-01-05 01:00:00	1
2020-01-05 02:00:20	0
...	...
2020-01-08 03:00:00	0
2020-01-08 04:00:00	1
2020-01-08 05:00:20	1
...	...
2020-01-09 21:00:00	0
2020-01-09 22:00:00	1

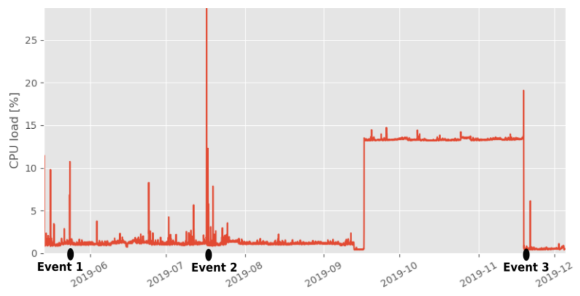
Now that all of the data available (metrics and events) is in a time-series format, the events can also be used as inputs, as if they are just another feature, and the ML can work just like for a time-series forecasting procedure. To predict one event, simply choose as the feature to "forecast", the respective event time-series.

2) Data normalization: The same normalization described in Section IV-A1 was applied to the metrics as inputs used in the event prediction. Given that the events time-series datasets already comprises values between $[0, 1]$, there is no need to normalize them.

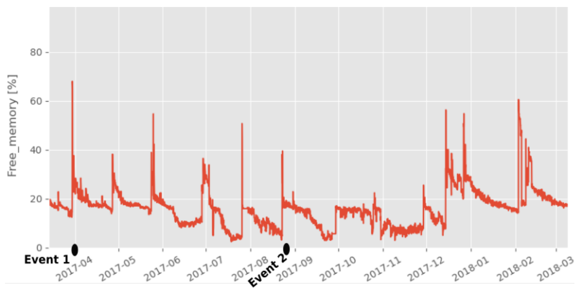
3) Sliding Window: The same sliding window approach described in Section IV-A2 was used with the addition of the event time-series. After testing all of the ML architectures with the different lengths, the results showed that input sequences longer than $20h$ did not improve any of the model's performances and that for output sequences longer than $4h$, the performance started to deteriorate notably. For output sequences until $4h$, the forecasts did not lower the accuracy significantly. As such, the results presented in Section V for event prediction were conducted with $N_{in} = 20; N_{out} = 4$.

4) Oscillations detection: After looking at the available metrics with graphical representations and crossing them with the respective event occurrences, it could be suggested that some of the metrics revealed destabilised behaviours in the times tightly close to an event occurrence. Such occasions are exemplified in Figure 7.

In an attempt to extract this information from the data and feed it directly to the network, an approach using standard deviations to capture those irregularities was used (every feature separately): for each point of each input sequence, compute the standard deviation of the last K values and use it as an extra input to feed the network. This way, each point of the input sequence would now be a 2D vector with: (a)



(a) CPU load.



(b) Free memory.

Fig. 7. Graphical representation of CPU load and free memory with event occurrences highlighted.

the actual value of the metric at the respective time-step and (b) the standard deviation of the previous K time-steps of the subject metric. This constant K used for the horizon of the standard deviation computations was deducted with a trial and error manual approach. The value that revealed best results was $k = 10$ and thus, is the one used for the results presented in section V.

Several other methods were experimented in place of the standard deviation, like the harmonic mean and a spike count similar to Section IV-A6, but none of them led to superior results.

5) Minority Class Oversampling: Given that the events evaluated in this work are derived from problems and failures, such occurrences only happen sporadically. This way, if the two possible outputs (0 and 1) are labelled into two classes – the minority class and the majority class –, the first would be for a prediction of 1 and the second for a prediction of 0.

To counter the majority class dominance, the minority class oversampling method was used. This method identifies the minority class occurrences in the training set and replicates them until the two classes are balanced enough for the model to learn to output both of the predictions. A value of 0.5 for the minority class oversampling technique ratio, $R_{OMC} = 0.5$, will result in a dataset where both classes have the same number of occurrence. Is this number is less than 0.5, the majority class will dominate by the correspondent ratio. If the oversampling ratio is more than 0.5, the (initially) minority class will dominate the dataset, by the respective ratio R_{OMC}.

6) Intelligent Methods Implementation: For the ML methods tested in the event prediction problems, the hyperparameter search was conducted like described in Section IV-A3 as well. The optimisation solver's conclusions were the same – the *adam* optimisation solver showed either equal or superior behaviour in all tests. However, since this is now a classification problem instead of a regression, the loss function used is the Binary Cross Entropy (BCE), which also seems to be a common choice for this type of predictions.

Similarly to the forecasting study, for event prediction, after several learning rates experimented between $[0.01, 0.1]$ ending up with the same results, the learning rate used in all final models was $lr = 0.05$. The activation functions, on the other hand, took a slight difference from the forecasting models – the output activation function (after the last layer) was always the sigmoid, which is the most common for problems where the output should be between $[0, 1]$. For the other activation functions, all of the ones presented in Section IV-A3 were tested, and again, the results were all very similar which makes this choice irrelevant as well.

7) Evaluation Metrics: For this type of problems, accuracy is not a good indicator of performance. A method that never predicts an event will probably have a high accuracy by merely predicting that the event never occurs.

This way, to evaluate the event prediction models, two measurements were used – precision and recall. Precision and recall can be mathematically described by

$$Precision = \frac{TP}{TP + FP} \times 100, \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \times 100 \tag{3}$$

where TP stands for true positives, FP stands for false negatives, and FN stands for false negatives.

V. Results and Discussion

A. Time-series Forecasting

The algorithms developed in this dissertation were put into production environments and tested across several different machines and with different metrics. For the results report, a set of data (collected from the same machine) was chosen to illustrate the performance of the embraced algorithms. The whole dataset accounts for three years of past data, and for the results presented in this chapter, it was split into a training set and test set for 80% and 20%, respectively.

The first metric is the free RAM memory of the machine, in percentage. The second metric is the CPU load of the machine, in percentage. The third metric is the rate of bytes per second that are being read from disk. The fourth metric is the free disk space, in GB. The fifth and last metric is the download speed of the machine, in Mbps. Out of the five metrics, the two that had more interest in being forecasted (product wise) were the free RAM memory and the CPU load. The results obtained in both of the forecasts were similar. This way, even though the free RAM memory will be the forecasted and evaluated metric in this chapter, the conclusions also apply to CPU load forecasts.

The ANN models were thoroughly tested for different numbers of hidden nodes and stacked layers. The rest of the hyperparameters are already defined in Section IV-A. For the ARIMA method, since the goal was to use it as the statistical reference to compare the ML models with, rather than optimize it thoroughly, a python module $auto\_arima()$ [12] was used to estimate the optimal parameters.

A table with the range of hidden nodes and stacked up layers tested in each model is present in Table V.

TABLE V
RANGES OF HYPERPARAMETERS EXPERIMENTED FOR THE FORECASTING MODELS.

Model	Nodes	Layers
FNN	[15, 100]	[1, 10]
Vanilla RNN variants	[15, 500]	[1, 5]
Encoder-decoder variants	[15, 150]	[1, 5]
Attention Encoder-decoder variants	[15, 150]	[1, 5]
Transformer	8	[1, 15]

The Transformer hidden nodes are referent to the parallel attention heads used in the multi-head concatenation (the same number used in the original work [8] was used).

Unfortunately, none of the models achieved satisfactory performances, and none of them accomplished forecasts good enough to be worth being used in a monitoring platform to assist in PdM. However, the configuration of each architecture that performed better (least bad), in terms of MSE, are written in Table VI.

TABLE VI
BEST PERFORMING FORECASTING MODELS AND CORRESPONDING MSE EVALUATIONS.

Model	Nodes	Layers	MSE_t	MSE_{t+2}	MSE_{t+5}
ARIMA	-	-	58.33	63.27	61.68
FNN	35-40-25	3	64.27	46.65	38.35
Vanilla RNN	40	1	11.53	13.12	14.70
Vanilla LSTM	50	1	6.61	9.75	11.89
Vanilla GRU	40	1	9.12	12.34	13.31
Enc-dec RNN	20	1	13.38	13.92	14.37
Enc-dec LSTM	20	1	13.47	13.91	14.73
Enc-dec GRU	20	1	13.64	13.25	14.86
Att. RNN	20	1	13.36	13.42	14.69
Att. LSTM	20	1	13.18	13.22	13.93
Att. GRU	20	1	13.43	13.27	14.70
Transformer	8	3	13.53	13.63	14.56
Naive	-	-	3.61	9.74	14.51

As the datasets are from trend tables, their collection period is of 1 hour, and the outputted forecast has an horizon of 6 hours. This way, the first (t), the middle $(t+2)$ and the last $(t+5)$ forecasts respective performances are present in Table VI. Furthermore, for comparison purposes, an extra row is appended with the results of the naive approach. The naive approach is an estimating technique in which the last period's values are used "blindlessly" as the next period's forecast, without adjusting them or attempting to establish causal factors. Despite the fact that, for real values that are more constant over time, the naive approach will output many forecasts with a low MSE , these forecasts are of no practical use because they will not add more information than the one already present in real-time monitoring. This way, for an intelligent model to be considered useful in this problem, its forecasts would not only have to compute forecasts with an MSE significantly below the naive approach but also detect abnormal behaviours before they occur.

Regardless of the bad results, some conclusions about what did the models learn from the data could be taken from their behaviour and graphical representations.

The "simplest" networks – the FNN – have fewer parameters to learn from and thus, in theory, are able to retain less information than the most robust ANN architectures. From their forecasts, it seems that the FNN were simply not able to model the training dataset and, as such, performed very poorly. However, it is notable that the last positions of the output sequence (the forecasts of further time-steps in the future) are less oscillatory than the first ones, which can be seen in Figure 8. This is the reason why in Table VI, the MSE of the forecast $(t+5)$ is smaller than the one of $(t+3)$.

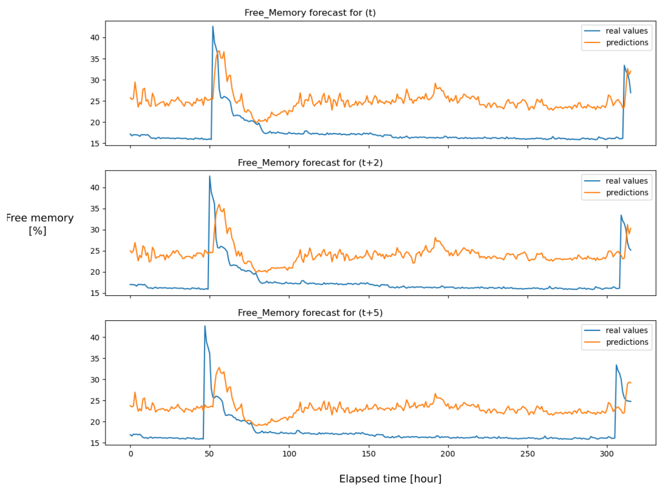


Fig. 8. Forecasts of the FNN model with 3 layers of 35, 40 and 25 hidden nodes, respectively.

The architecture that best performed in terms of MSE was the vanilla LSTM, and it did so because it managed to learn something close to a naive approach, and not by "intelligently" computing forecasts. A sample of the forecasts computed by the "winner" vanilla LSTM is graphically represented in Figure 9.

It can be deducted from Figure 9 that the network learned to output a constant for most of the time and that when it detects a steep oscillation, it reacts with a softened naive approach – softened in the sense that, above a certain amplitude, it replicates the oscillation but with a less steep reaction. It is possible that the LSTM learns that by applying a naive approach, it will achieve a smaller MSE (which is the mathematical goal of the training process). Moreover, an LSTM network with more nodes can learn more information and is able to learn that by outputting a constant value and softening the oscillations reaction, the MSE can be even smaller. Furthermore, the most oversized LSTM networks tested in this work went even further and learned to just output a constant regardless of the input sequence. As an example, a forecast of a LSTM network

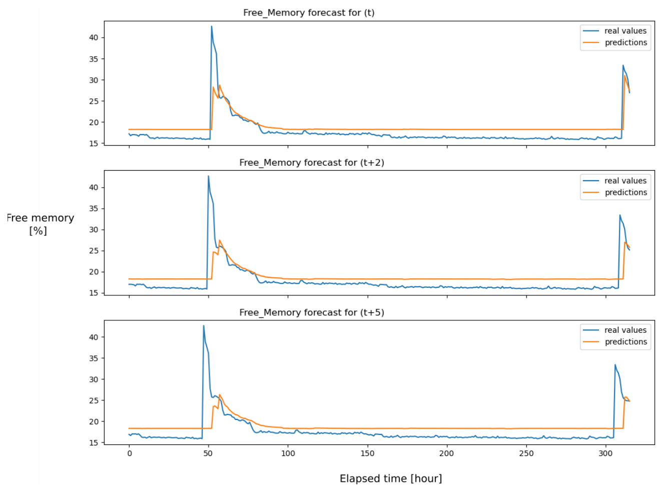
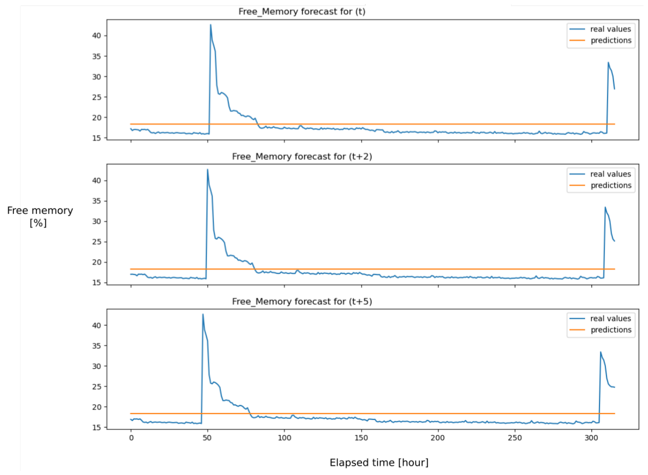
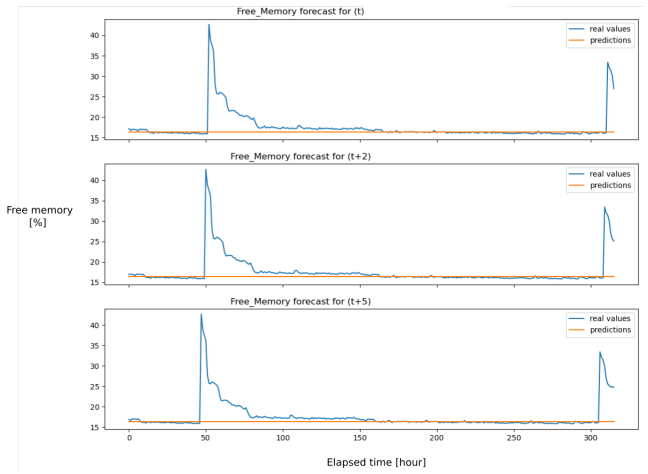


Fig. 9. Forecasts of the vanilla LSTM model with 50 hidden nodes and 1 single layer.



(a) Forecasts of the vanilla LSTM model with 3 layers of 200, 500 and 50 hidden nodes, respectively.



(b) Forecasts of the Encoder-decoder LSTM model with 20 hidden nodes and 1 single layer.

Fig. 10. Constant forecasts computed by the larger and more robust architectures.

with 3 stacked up layers of 200, 500 and 50 hidden nodes, respectively, is represented in Figure 10a.

Just like the biggest vanilla LSTM models, the larger and more complex architectures – Encoder-Decoder, Encoder-Decoder with Attention and the Transformer – learned to forecast a constant output regardless of the inputs as well, even with one single layer and few nodes (20). When increasing the number of hidden nodes and the number of stacked layers, these last models show the exact same behaviour. Which probably means that the most information that can be taken out of the training data is that a constant forecast is the one that will result in a lower MSE. As a representative example of these models, in Figure 10b is a graphical representation of the forecasts computed by an encoder-decoder LSTM model with 20 hidden nodes and 1 layer.

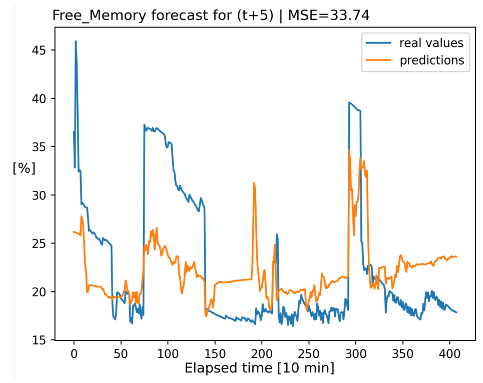
In terms of output shapes (oscillations, naive approaches and constant outputs), the RNN cells of the three types studied all reveal similar results. This way, all of the graphical examples given for architectures with LSTM cells can be generalised for the Elman RNN and the GRU.

1) Period Reduction Trial: The metrics used for this experiment were the same ones, but the with specifications described in Section IV-A6.

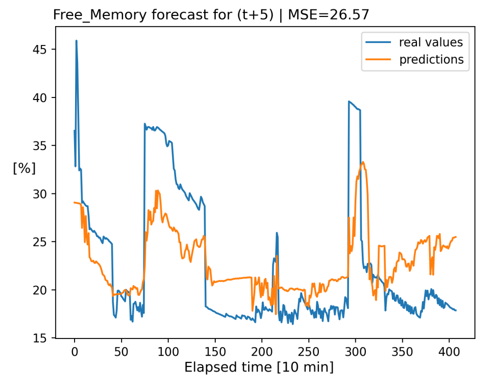
With the heavier architectures – Encoder-Decoder, Encoder-Decoder with Attention and the Transformer – for the most part of the (repeated) experiments, the models outputted constant values, similar to the ones obtained in Figure 10b. As such, either these models ignored the spikes count as an extra feature, or what they learned from the spikes simply reinforced what had been learned from the plain data – outputting a constant will likely minimize the loss function (MSE) across new datasets.

For the FNN and vanilla RNN variants, the period reduction trial revealed visible differences. Regarding the evaluation metric used (MSE), for the RNN architectures some tests led to better results than the ones in Table VI and others did not, given that different training iterations constantly led to different results. For the FNN however, with 2 hidden layers of 45 and 75 hidden nodes, respectively, the results were slightly better (even considering the inconsistent repetitions) – with MSE values in between $[20, 35]$ for the last point of the output sequence (the 6th), for the free RAM memory forecast. Nonetheless, given that the time-steps of the sequences in this experiment are 10min apart (as described in Section IV-A6), the last step predicted $(t + 5)$, is only 1 hour in the future instead of 6. This way, it is hard and inaccurate to compare these values with the ones from Table VI.

Moreover, it is notable that the FNN in this period reduction trial approach, reacts steeply to spikes in data and, at times, is able to detect them ahead of time, although still with high MSE values overall. Furthermore, it was evident that the threshold T defined to consider the spikes highly influences the “sensitivity” towards spikes. In Figure 11 is a comparison of a FNN trained with a threshold $T = 7$ – Figure 11a; and the same FNN trained with $T = 15$ Figure 11b. A lower threshold will count more spikes, as it will require a lower amplitude to be considered and counted as a spike. And vice-versa for a higher threshold.



(a) FNN model forecast of free RAM memory with a spike detection threshold of $T = 7$.



(b) FNN model forecast of free RAM memory with a spike detection threshold of $T = 15$.

Fig. 11. Graphical comparison of the forecasts over different threshold values for the period reduction trial.

From Figure 11a it is clear that the network was able to forecast steep oscillations (although not with the correct amplitude). However, it also does forecast some oscillations shortly before time (for example, around the time-step 230), and even some oscillations that did not take place at all (for example, around the time-step 190). In Figure 11b, with a less sensitive threshold line, the model does not forecast steep oscillations so easily and instead tries to follow the metric's previous values. Although this second one does not seem to add any valuable forecasting information, it does result in a lower MSE – 26.57, which is significantly lower than 33.74 for Figure 11a.

B. Event Prediction

To illustrate the behaviour and performance of the algorithms developed for event prediction, the set of input metrics described in Section V-A was used alongside two different events S_1 and S_2 that the models learned do predict. These events are related to problems in IT systems that are useful to be predicted in advance of the actual occurrence. Both S_1 and S_2 are referent to services running on the same machine where the metrics were collected, service 1 and service 2, respectively. Whenever service 1 crashes or is down, the event S_1 is triggered, and when it is back up and running, the event S_1 is signalled and reverts the trigger, producing this way a dataset of the form of Table III. For service 2, the same mechanism is used with the correspondent event S_2. For the event prediction problems, the dataset split into training set and test set was also of 80% and 20%, respectively. To have a clearer visualization of the occurrence of events alongside IT systems, one can look at the example of Figure 7.

The ANN models studied were also experimented for a vast range of hidden nodes and stacked up layers, following the implementation techniques described in Section IV-B. In Table VII is a summary of the intervals tested for each architecture.

TABLE VII
RANGES OF HYPERPARAMETERS EXPERIMENTED FOR THE EVENT PREDICTION MODELS.

Model	Nodes	Layers
FNN	[15, 100]	[1, 6]
Vanilla RNN variants	[15, 500]	[1, 3]
Encoder-decoder variants	[15, 150]	[1, 3]
Attention Encoder-decoder variants	[15, 150]	[1, 3]
Transformer	8	[1, 10]

The best performing configurations for each model are present in Table VIII, alongside the correspondent evaluation metrics described in Section IV-B7 – precision (P) and recall (R).

TABLE VIII
BEST PERFORMING CONFIGURATIONS FOR THE EVENT PREDICTION MODELS – WITH THE CORRESPONDING PRECISION AND RECALL.

Model	Nodes	Layers	S_1		S_2	
			P	R	P	R
FNN	45-60	2	48.6	56.0	47.8	58.9
Vanilla RNN	35	1	36.7	35.5	33.7	34.8
Vanilla LSTM	25	1	43.8	44.2	45.1	51.0
Vanilla GRU	30	1	40.0	41.6	44.8	44.4
Enc-dec RNN	25	1	33.6	35.3	31.3	35.4
Enc-dec LSTM	20	1	36.4	34.8	35.2	35.7
Enc-dec GRU	15	1	34.2	34.2	36.8	32.2
Att. RNN	30	1	37.2	32.9	34.1	34.5
Att. LSTM	25	1	41.8	32.9	38.2	37.6
Att. GRU	30	1	38.1	31.1	36.1	37.3
Transformer	8	6	45.3	40.7	48.3	41.0

The event prediction algorithms' results are not ideal, but already provide useful outputs and show that the algorithms were able to learn from the input data on how to predict events. The FNN is the undisputed winner with the techniques developed for these problems, in both precision and recall measurements. The vanilla LSTM and GRU networks also performed reasonably and, surprisingly, the Transformer outperformed all of the other encoder-decoder models.

Given that the oscillation detections (described in Section IV-B) were the key for the models to learn to predict the event occurrences, the FNN managed to take better advantage of them due to the fully connected layers that compose its architecture. The RNN variants, on the other hand, process the inputs sequentially, which attenuates the presence of the spikes over the network. The fact that the encoder-decoder architectures performed bellow the vanilla RNN only strengthens this hypothesis, given that the inputs have to go through two entire RNNs. The attention mechanism still caught up some information that faded through the encoder-decoder but still underperformed the vanilla RNN models. For last, the Transformer does not have the feedback loop of all the RNN cells based models and thus is able to better capture the oscillations information from the inputs. Nonetheless, it still underperformed the integral fully connected layers of the FNN model, that are able to interpret the oscillation detection inputs and directly forward it to predict the events.

Moreover, the minority class oversampling technique used for all the above results was implemented with a ratio of $R_{OMC} = 0.5$. This parameter highly influences the outputs and might be important to adjust it according to the programmer's preferences and needs. If the ratio is increased, the model will be fed with more event occurrences, which will result in a lower precision but a higher recall. On the other hand, if the ratio R_{MOC} is decreased, the model will be fed with fewer event occurrences, which will result in the opposite – higher precision and lower recall. In Table IX are illustrative results of experiments with difference ratio R_{MOC} values by the best performing model – FNN.

TABLE IX
R_{OMC} VARIATIONS FOR EVENT PREDICTIONS WITH THE FNN MODEL OF TABLE VIII.

R_{OMC}	S_1		S_2	
	P	**R**	**P**	**R**
0.25	70.34	37.05	68.43	39.72
0.5	48.55	55.95	47.75	58.94
0.75	39.08	71.21	38.30	75.73

VI. Conclusion

A. Time-series forecasting

The more advanced and complex models – vanilla encoder-decoders, encoder-decoders with the attention mechanism, and Transformers – could not outperform the vanilla RNN models, namely the LSTM which is the model that scored the lowest MSE in tests. Unfortunately, none of the models tested in this work delivered encouraging results. The reason could have either been that the algorithms were not suitable to model the data, or that the data itself is not capable of establishing causal factors. Consequently, each in their own way, the models ended up learning naive approaches or to just output a constant in order to minimize the loss function, regardless of the "reasonless" outputs. Moreover, given the disappointing results in this section, an alternative data preprocessing/rearrangement technique was employed (described in Section IV-A6), to try to make sense of spikes present in the data and maybe justify the bad results. Moreover, this experiment was not intended to be put into production, as it was incompatible with the monitoring platform where the algorithms were deployed. However, due to the scarce data needed for this experiment, solid conclusions could not be taken. Nonetheless, the spikes counting technique helped the FNNs to forecast steep oscillations in data, that could not have been done before (even though these forecasts were unstable).

B. Event Prediction using time-series

A visual data analysis of the problem recognized that often event occurrences were surrounded by data oscillations on the metrics collected on the same machine as the events. Accordingly, a method to detect these oscillations (with standard deviation) and explicitly feed that information to the model was developed (described in Section IV-B4). The oscillations detection approach significantly improved the predictions, that before this treatment had very poor performances. As reported in Section V-B, the predictions are not ideal, but their outputs can be useful and are already capable of being interpreted. The FNN is the model that revealed the best performances, as it can take batter advantage of the oscillation detection data thanks to the fully connected layers that composes its architecture. The recurrent networks end up attenuating the oscillation detection inputs in their feedback loops, resulting in worse results than the simpler FNN. The Transformer does not comprise any recurrent mechanisms and thus is able to forward more easily the oscillation detection inputs to the outputs of the network. Even though still behind the FNN, the Transformer outperformed the other methods based in recurrent units. Lastly, the oversampling minority class technique can be manipulated, by tuning the ratio R_{OMC} as desired, which will result in projecting a deliberate bias in the predictions, to either be riskier or more conservative (complete explanation in Section V-B).

References

[1] (2019) Importance and benefits of predictive and preventive maintenance. [Online]. Available: https://www.eaglecmms.com/importance-and-benefits-of-predictive-and-preventive-maintenance/

[2] (2018) State of deep learning: Industrial PdM | LinkedIn. [Online]. Available: https://www.linkedin.com/pulse/deep-learning-iiot-checklist-lothar-schubert/

[3] J. L. Elman, "Finding structure in time," vol. 14, no. 2, pp. 179–211, 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/036402139090002E

[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory," vol. 9, pp. 1735–80, 1997.

[5] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[6] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014. [Online]. Available: http://arxiv.org/abs/1409.3215

[7] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2016. [Online]. Available: http://arxiv.org/abs/1409.0473

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[9] N. Qian, "On the momentum term in gradient descent learning algorithms," vol. 12, no. 1, pp. 145–151, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608098001166

[10] Neural networks and deep learning. [Online]. Available: https://www.coursera.org/learn/neural-networks-deep-learning

[11] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. [Online]. Available: http://arxiv.org/abs/1412.6980

[12] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," p. 25, 2012.