A Comparative Study of State-of-the-Art Machine Learning Algorithms for Predictive Maintenance

Conference Paper · December 2019

DOI: 10.1109/SSCI44817.2019.9003044

3 authors, including:

Luis P. Silvestrin
Vrije Universiteit Amsterdam
4 PUBLICATIONS **22** CITATIONS
SEE PROFILE

Ger Koole
Vrije Universiteit Amsterdam
234 PUBLICATIONS **7,489** CITATIONS
SEE PROFILE

A Comparative Study of State-of-the-Art Machine Learning Algorithms for Predictive Maintenance

Luis P. Silvestrin
Vrije Universiteit Amsterdam
Faculty of Science
Dpt. of Computer Science & Dpt. of Mathematics
Amsterdam, the Netherlands
l.p.silvestrin@vu.nl

Mark Hoogendoorn
Vrije Universiteit Amsterdam
Faculty of Science
Department of Computer Science
Amsterdam, the Netherlands
m.hoogendoorn@vu.nl

Ger Koole
Vrije Universiteit Amsterdam
Faculty of Science
Department of Mathematics
Amsterdam, the Netherlands
ger.koole@vu.nl

***Abstract*—Predictive maintenance strives to maximize the availability of engineering systems. Over the last decade, machine learning has started to play a pivotal role in the domain to predict failures in machines and thus contribute to predictive maintenance. Ample approaches have been proposed to exploit machine learning based on sensory data obtained from engineering systems. Traditionally, these were based on feature engineering from the data followed by the application of a traditional machine learning algorithm. Recently, also deep learning approaches that are able to extract the features automatically have been utilized (including LSTMs and Convolutional Neural Networks), showing promising results. However, deep learning approaches need a substantial amount of data to be effective. Also, novel developments in deep learning architectures for time series have not been applied to predictive maintenance so far. In this paper, we compare a variety of different traditional machine learning and deep learning approaches to a representative (and modestly sized) predictive maintenance dataset and study their differences. In the deep learning approaches we include a novel approach that has not been tested for predictive maintenance yet: the temporal convolutional neural network. We compare the approaches over different sizes of the training dataset. The results show that, when the data is scarce, the temporal convolutional network performs better than the common deep learning approaches applied to predictive maintenance. However, it does not beat the more traditional feature engineering based approaches.**

***Index Terms*—predictive maintenance, machine learning, temporal convolutional networks**

I. Introduction

Predictive maintenance (PdM) is a growing alternative to traditional methods such as breakdown corrective maintenance or scheduled predictive maintenance for several industries, used the automotive industry, aerospace and many other fields [1] [2]. It can maximize the operational availability of engineering systems and prevent downtime due to unscheduled maintenance, reducing the costs and increasing safety. There are two classes of PdM solutions: model-based (physics-based models) and data-driven (machine learning models) [3]. While the model-based solutions require extensive domain knowledge, which is not always available, data-driven approaches try to learn predictive models from the data automatically, making them useful for a wide range of PdM problems.

Among the data-driven approaches for PdM, there are complex machine learning models that require large amounts of data, such as deep learning, and simpler methods that can perform well on small datasets, but usually require some feature engineering. Many deep learning solutions have been proposed for PdM problems such as remaining useful life estimation of wind turbines [4] or the degradation assessment of rolling bearings [5]. These show promising results compared to other state-of-the-art techniques. However, deep learning techniques are known for requiring large amounts of labeled data to be trained properly, and, in many real-life PdM problems, such big datasets are not available or the prediction targets are too unbalanced (some labels have much less occurrences than others). In this work, we investigate how well deep learning approaches to PdM, such as the Long Short-Term Memory networks (LSTM) [6], perform in scarce sensor data scenarios, and compare them to simpler machine learning models with feature engineering, such as a random forest and a decision tree. We also add to this comparison a recently proposed deep learning architecture, the Temporal Convolutional Network (TCN) [7], which has not yet been evaluated in a real-life PdM problem. This is a new kind of comparison, given that previous works only show the performance of deep learning on large or simulated datasets [4] [8], or do not include any kind of deep learning algorithm [2].

To evaluate the performance of the machine learning techniques in a real-life PdM scenario, we use the hydraulic system condition monitoring dataset [9]. We train the algorithms with training sets of varying sizes. For the deep learning algorithms we do not use feature engineering. As alternative we study a random forest [10], decision tree (see e.g. [11]), and K-nearest neighbors [12] combined with simple (time and frequency-based) features extracted from the sensors. We compare their performance on a completely set aside test set of fixed size. Finally, we compare the accuracy curves obtained on the test data for each algorithm against the training data size to compare how the training set size affects their performance.

The remainder of this paper is structured as follows: Section II presents other existing Machine Learning approaches for predictive maintenance. Section III explains the different machine learning methods that have been exploited, including a more elaborate explanation of the TCN architecture. Section IV details the dataset and set up of the experiments and Section

V shows the results obtained. Finally, Section VI discusses the findings of this work.

II. Related Work

Recently, many papers have tackled PdM problems using deep learning approaches. There are works that developed LSTMs [8] and Convolutional Neural Networks (CNN) [4] architectures to predict the remaining useful life of wind turbines, but they do not provide any analysis about the impact of the training set size on their performance. There is also work which shows how to assess the degradation of rolling bearings using LSTMs [5], but compares it only with other deep learning models, without taking simpler Machine Learning algorithms into account. Of all the related papers, however, there are still no experiments using the TCN topology, which makes this paper the first to test it in the PdM setting.

Other comparison paper of machine learning techniques for predictive maintenance ([1]–[3]) do a comparison with many machine learning algorithms for PdM tasks, but do not include deep learning models, neither do they show the efficiency of the techniques on scarce data. Considering the hydraulic system dataset used in this work, [9] does a comparison between different Machine Learning methods, but all the techniques it presents are based on feature engineering and they do not consider deep learning in their experiments.

III. Methodology

In this section, we first explain on a high level how we set up and train the different machine learning models given the raw dataset we have. Thereafter, we go into more detail on the individual algorithms including the LSTM and the TCN topologies, their structure, advantages, and disadvantages.

A. Process

Figure 1 shows the process we use to apply the different machine learning techniques for predictive maintenance. We start with a PdM dataset, which typically captures a variety of sensors measured at different frequencies. Let us assume that we have p sensor values that are measured over time. Given that all our approaches assume an equally spaced time line without a huge amount of missing values we bring this original dataset back to one single sampling frequency Δt. Δt should strike a balance between not losing too much useful fine-grained information and not suffering from too many missing values.

To create values at each step (i.e., over each interval of length Δt), we summarize the signals using simple statistics. From each time window, we extract 4 statistical features, the mean, standard deviation, and the minimum and maximum values, which are commonly used in signal processing tasks and are fast to compute (see Figure 2). We have now obtained the dataset we are going to work with, including $4p$ features. Once the statistical features are extracted, they go through a normalization process. This is done feature-wise by computing the mean and standard deviation over all time points of all the examples in the data. Then each time point value has the mean subtracted and is divided by the standard deviation.

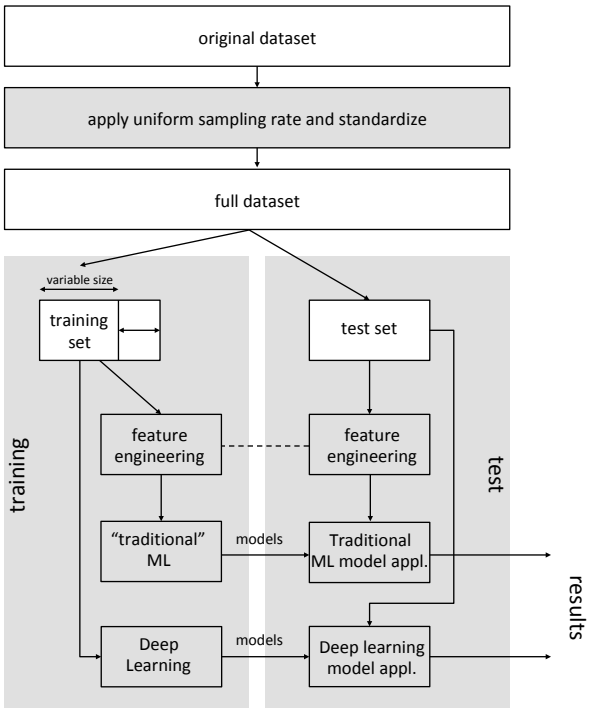

Fig. 1. Process for different learning algorithms.

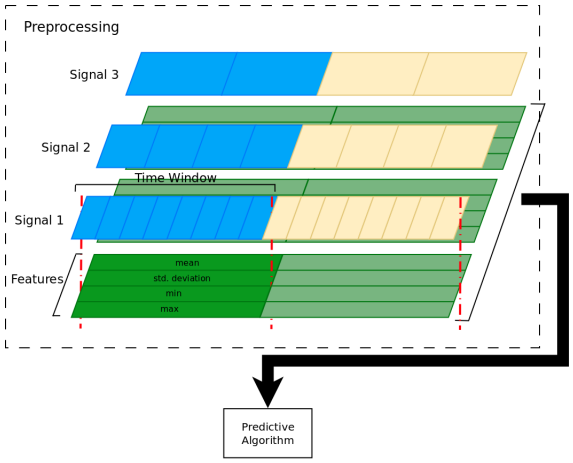

Fig. 2. Summarization of signals with different sampling rates using a time window.

We then identify a training and test portion in the data (where the test portion is fixed) and vary the training portion provided to train the algorithms. For the deep learning approaches we directly learn from the training portion of the data (see Section IV for details on how we optimize the hyperparameters), resulting in a model that we apply to the test set. For the other algorithms, we first apply feature engineering and then apply the algorithms and follow a similar process as we have described for the deep learning thereafter. Let us now consider the specifics of each learning algorithm in more

detail.

B. Long Short-term Memory

The long short-term memory architecture [6] is a deep learning topology that reached great success in language translation and modeling tasks in recent years by its ability to handle long temporal sequences. It is an extension of the Recurrent Neural Network topology which uses gate cells to select the information that goes into its hidden state, increasing the capacity of "remembering" past information and attenuating the problem of unstable gradients during training. In this architecture, there are three modulating gates: the forget, input and output gates. The forget gate selects which parts of the current state go through the next state, the input gate, modulates the current input before it is added to the new state, and the output gate modulates the values that pass from the current state to the output. A more thorough explanation of the LSTM topology can be found in [6].

In the PdM domain, many works are showing the advantages of the LSTM compared to other Machine Learning models [5] [8]. For these reasons, we decided to include the LSTM in our experiments. In order to apply it to our dataset, we create $4p$ inputs and feed the network the series of values of a set length. As output we take the number of classes as part of the dataset (or 1 in case of 2 classes).

C. Temporal Convolutional Networks

LSTMs have been the state-of-the-art deep learning topology for time-series problems such as speech recognition or natural language processing. However, a new deep architecture, the Temporal Convolutional Network [7] was presented recently showing better results than the LSTM in many classic sequence modeling problems, such as language modeling or music modeling. In these tasks, the TCN has obtained a better performance with fewer parameters.

Temporal convolutional networks are based on the idea of causal convolutions based on a complete time series as input. The causal convolutions consist of convolutions with 1D filters with input padding, in a way that the input size is preserved, hence allowing sequence-to-sequence mappings. The causal convolutions guarantee that the output at each time position t is a function exclusively of the inputs before t, so it cannot predict a value at t based on future inputs. The TCN can efficiently process long sequences by using dilation in its convolutions, increasing the receptive field exponentially proportional to the number of stacked convolutions in the topology. A dilated convolution can be expressed as:

$$F(t) = (\mathbf{x} *_d f)(t) = \sum_{i=0}^{k-1} f(i) \cdot \mathbf{x}_{t-d \cdot i} \qquad (1)$$

Here, we take the t-th time step of the convolution of an time-series input $\mathbf{x}$ convolved by a filter $f : \{1..k\} \longrightarrow \mathbb{R}$. On each step t we make an element-wise multiplication between the filter and the k previous elements of $\mathbf{x}$. The dilation factor d is the number of steps in time that are skipped between each of the k elements of $\mathbf{x}$ involved in the convolution up to t. In the TCN, dilated convolutions are stacked on top of each other, and the dilation factor is multiplied at each level (see Figure 3). This way, in the higher levels, the convolutions can reach far in the past, making this topology able to process long sequential data with few parameters.

Unlike recurrent topologies such as the LSTM, that need to process the input sequentially, the TCN allows parallelism since it is based on simple convolutions, accelerating the training and prediction processes. Another advantage is the parameter sharing of the convolutions, which allows the TCN to efficiently learn how to extract features from the sensor readings. For these reasons, we decided to try this new topology in a Predictive Maintenance use case, which has never been done before.

The series fed to the input of the TCN are set to the same length as we set for the LSTM network. As output we have a number of neurons equal to the number of output classes.

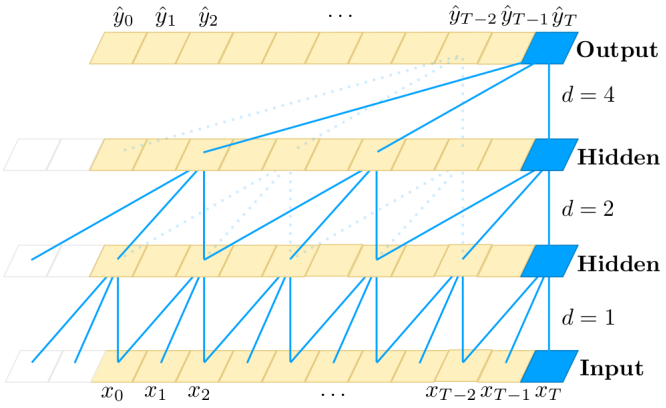

Fig. 3. Example of three stacked dilated convolutions. At every hidden layer, the dilation factor is multiplied by 2, doubling the range of the convolution in time and adding only 3 extra parameters (taken from [7]).

D. Traditional Machine Learning Algorithms

In addition to the deep learning architectures, there are fundamental Machine Learning algorithms that are used extensively in the industry. They are known for their simplicity, having fewer parameters and hyperparameters to tune, and also requiring less data to train compared to deep learning models. Furthermore, they are often more insightful. From those models, we selected three that are representative for different types of algorithms: a decision tree, random forest, and K-nearest neighbors, to compare their performance with the deep learning algorithms and to use as a baseline for the scarce data case.

The algorithms were not developed to work with sequential data. In order to apply them to PdM problems, we need to perform feature engineering (as already mentioned before). In this case, we opt for features that reduce the time-series into a single statistical feature. This can be done in a similar way as was shown previously for making the sampling rate more uniform, except now we use a time window as long as the

input sequence. Once this is done, the resulting data does not have the temporal dimension anymore, so it can be represented as a matrix where each row is an example and each column is a statistical feature extracted from a sensor in the input.

IV. Experiments

In this section, we detail the experimental setting, including the data used, and the parameters used for each algorithm. We also explain how the details about the training and testing of the models.

A. Hydraulic System Sensor Dataset

The Hydraulic System Sensor dataset [9] is composed of measurements from a collection of sensors installed in a hydraulic test rig. There are 6 sensors for pressure (*PS*), 4 for temperature (*TS*), 2 for volume flow (*FS*), one for vibration (*VS*), one for cooling efficiency (*CE*), one for cooling power (*CP*) and the efficiency factor (*SE*). The measurement rates are 100 Hz for *PS*, 10 Hz for *FS* and 1 Hz for all the others. The prediction target is the *internal pump leakage*, which can be labeled as *no leakage*, *weak leakage* and *severe leakage*, resulting in three distinct classes. This label is annotated for every cycle of 1 minute, and there are 2205 cycles in total. Thus, for each of those cycles, we summarize the sensor measurements using 4 simple statistical features: minimum and maximum values, mean, and standard deviation, as described in Section III-A. After this step, we end up with 64 features (4 features per sensor) per cycle. Figure 4 illustrates how each class spreads over some of the different features. For TCN and LSTM, we use a time window Δt of 6 seconds, so each of the features is summarized in 10 time steps.

B. Training and Testing

To evaluate the algorithms under the scenario of scarce data, we trained them using training sets of varied sizes and tested them on a test set of fixed size. The test set is composed of 15% of the data, while the rest is used for training.

Each algorithm is trained until convergence on a sample of the training set, tested on the test set, and then we report the classification error obtained on both training and test sets. The training sample is drawn randomly from the training dataset. We start with a sample of 200 examples and repeat this process iteratively, each time incrementing the size of the random sample by 200, up to 1600 examples. Per training set size we take 16 draws and on each draw run our algorithms. This allows us to tackle the stochasticity of both the drawing of the random sample as well as the stochasticity of the learning algorithms.

We have optimized the hyperparameters of the different algorithms based on rigorous experimentation on the training set. The TCN and the LSTM are both trained using the optimization algorithm Adam [13] with an initial learning rate of 0.01 for 200 epochs. The objective function used for both is the cross-entropy. The LSTM model is composed of 16 hidden units, and its output goes through a fully connected layer that outputs 3 values as the probability of belonging to each class.

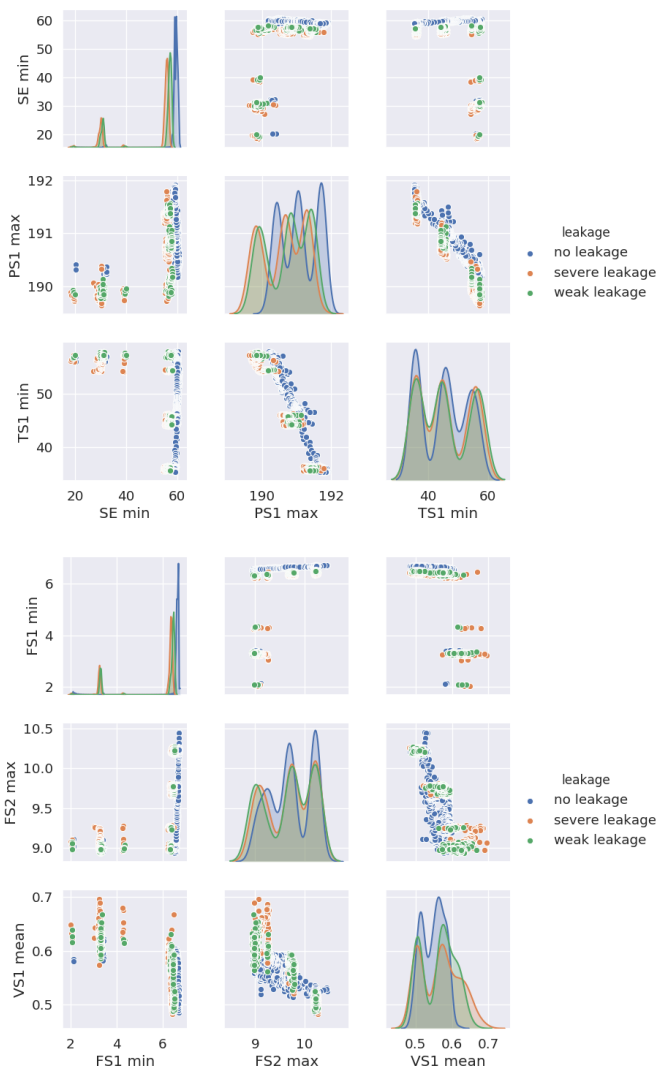

Fig. 4. An overview of six of the features extracted from the hydraulic system dataset.

Finally, the three values go through a softmax activation to produce the final output. The topology chosen for the TCN uses 8 filters of 3 stacked dilated convolutions, with dilations 1, 2 and 4, and kernels of size 2. On top of that, there is a fully connected layer producing the class outputs followed by a softmax, similarly to what was done for the LSTM.

For the decision tree, we used the Gini impurity as the criterion to select the splitting attribute for each node. For the random forest, 10 decision trees were built using that same criterion. The k-nearest neighbors were trained using $k = 1$, turning it into a nearest neighbor classifier.

The algorithms were compared based on the classification error obtained on the test set. This metric is defined as 1 minus the accuracy, which is the sum over all the correctly classified examples divided by the total number of examples (see Equation 2).

$$e = 1 - \frac{1}{N}\sum_{i=0}^{N} \mathbb{1}_{\hat{y}_i, y_i} \tag{2}$$

V. Results

In this section, we present the results obtained from the experiments described previously, with the goal of assessing the performance of the TCN and the LSTM in a real-life PdM use case with fairly limited data compared to approaches that use feature engineering. We first present the comparative results. Thereafter, we focus more in depth on the learning behavior of the individual learning algorithms.

A. Comparative Results

First, we show the results obtained with the traditional machine learning algorithms, that will serve as a baseline for the deep learning methods. The decision tree, random forest and k-nearest neighbors are referred as DT, RF and KNN respectively in Figure 5. The random forest, an ensemble of decision trees, shows a lower error than a single tree in all cases, as expected. It has similar performance as the k-nearest neighbors, both having around 3% error when trained with only 400 examples. When comparing the performance on the training and test set, we see that all algorithms have a nearly zero error on the training set right from the start (i.e., with a small size of the training set) while the performance on the test set is much worse but gradually improves as more data is used for training. These results are to be expected based on learning theory (see, e.g., [14]). When the training set size increases we also see that the variability of the results reduces. Finally, we see that the DT is most prone to overfitting.

The results of the deep learning approaches are shown in Figure 6. The TCN and the LSTM present a higher classification error compared to the baseline machine learning algorithms, especially when trained with small datasets. The LSTM had the worst performance: around 40% error in all the cases, it does not seem to have enough data to learn from and recognize patterns despite the simple architecture used (more complex architectures yielded even worse results). We see that the difference in performance between the training and test is small, mainly caused by regularization. The TCN shows a very different behavior. While it is still outperformed by the traditional machine learning algorithms, we do see its error steadily decreases when it is presented with more training data. We believe that with slightly more data it would be able to outperform the traditional machine learning approaches (but do not have that available in the selected dataset). We also see the variability of the results decreasing rapidly (which is initially much higher than that of the LSTM). This all can be explained by the lower number of weights that need to be learned.

Finally, we put side by side the performances obtained by all the tested algorithms (see Table I). To summarize, the decision tree, random forest and k-nearest neighbors showed a more consistent learning over all the runs on the small datasets in comparison to the TCN and the LSTM. The TCN

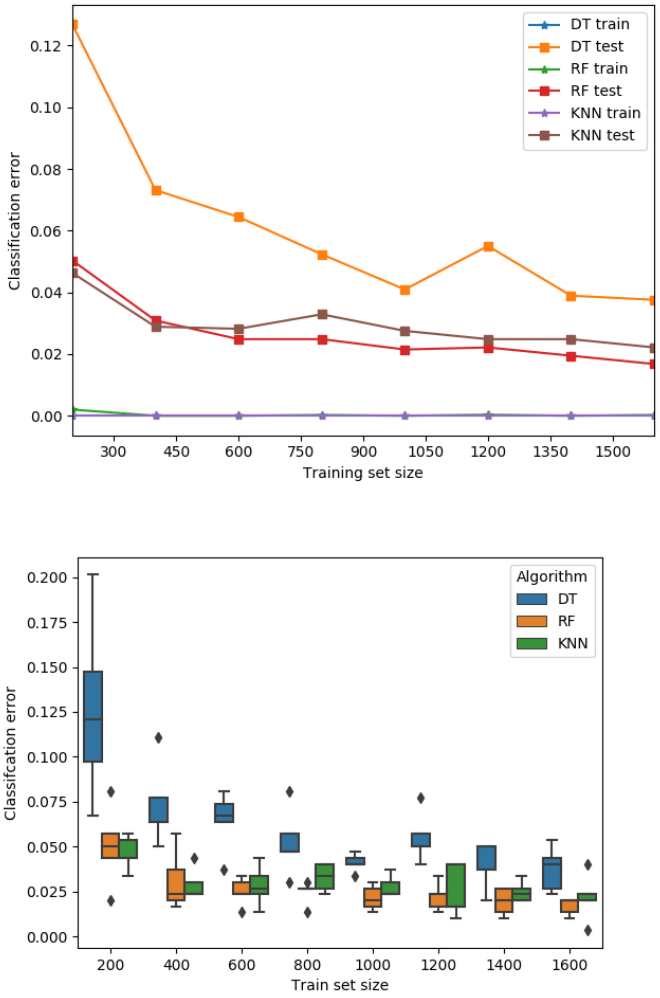

Fig. 5. The reported classification error on the test set for the other machine learning algorithms after training with datasets of varied sizes. On top, the line plot showing the slightly decreasing trend of the error as the training set size increases. On bottom, the variance of the error obtained from 16 runs, each time with different random samples for the training data.

shows an oscilating performance when the training set is small, but slowly approaches the baseline methods as training data increases. On the other hand, the LSTM does not show any improvement even when more data is provided.

B. In-Depth Analysis

To bring some insights into the training of the deep learning approaches, we compare the learning curves obtained using the full training set over the 200 epochs we use for training. The learning curve of the LSTM shows that it learns very slowly and that it requires more epochs to be trained in comparison to the TCN. It is visible that the validation loss is smaller than the train loss, and can be explained by the effect of dropout during the training. We can observe that the TCN converges much faster than the LSTM, reaching an optimal training loss in the final epochs. The validation loss oscillates during the training, but in the final epochs it stabilizes, showing that the

TABLE I

THE MEAN AND STANDARD DEVIATION OF THE CLASSIFICATION ERROR ON THE TEST SET OBTAINED BY EACH ALGORITHM WITH VARYING TRAINING SET SIZE.

Size	DT	KNN	RF	TCN	LSTM
200	0.11 ± 0.04	0.05 ± 0.02	0.06 ± 0.02	0.46 ± 0.13	0.42 ± 0.02
400	0.07 ± 0.03	0.04 ± 0.01	0.04 ± 0.02	0.39 ± 0.21	0.39 ± 0.03
600	0.07 ± 0.02	0.03 ± 0.01	0.03 ± 0.01	0.34 ± 0.23	0.40 ± 0.02
800	0.05 ± 0.01	0.03 ± 0.01	0.03 ± 0.01	0.22 ± 0.12	0.40 ± 0.02
1000	0.04 ± 0.01	0.03 ± 0.01	0.02 ± 0.01	0.15 ± 0.15	0.40 ± 0.03
1200	0.05 ± 0.02	0.03 ± 0.01	0.02 ± 0.01	0.08 ± 0.09	0.40 ± 0.02
1400	0.04 ± 0.01	0.02 ± 0.01	0.02 ± 0.01	0.06 ± 0.05	0.39 ± 0.04
1600	0.04 ± 0.01	0.02 ± 0.01	0.02 ± 0.01	0.12 ± 0.21	0.40 ± 0.03

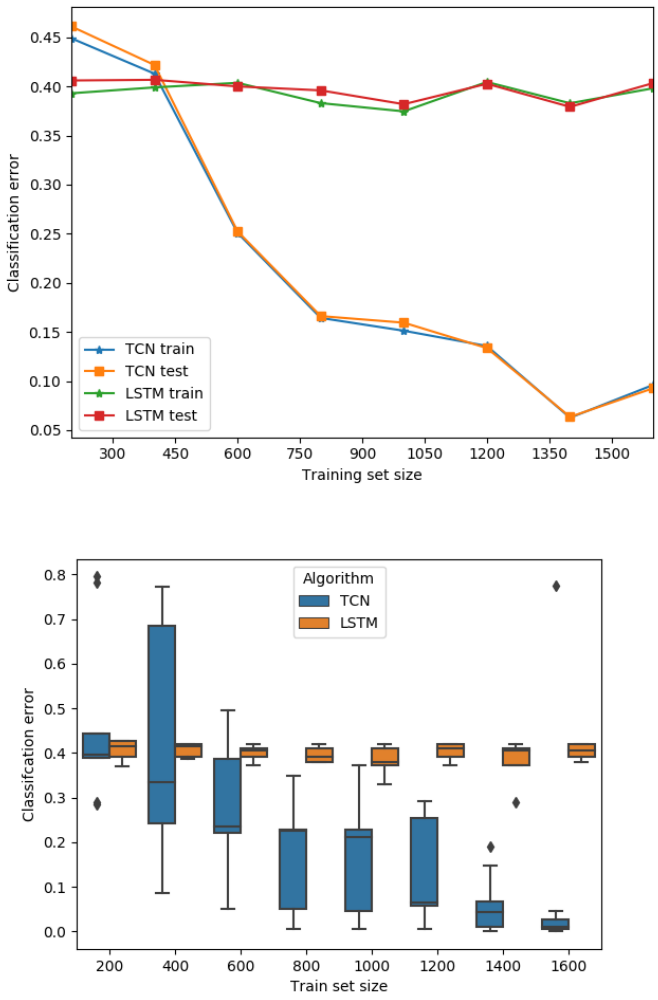

Fig. 6. Classification error obtained from the experiment with TCN and LSTM (note the different y-scale compared to Figure 5)

TCN becomes able to generalize also in the validation set (see Figure 7).

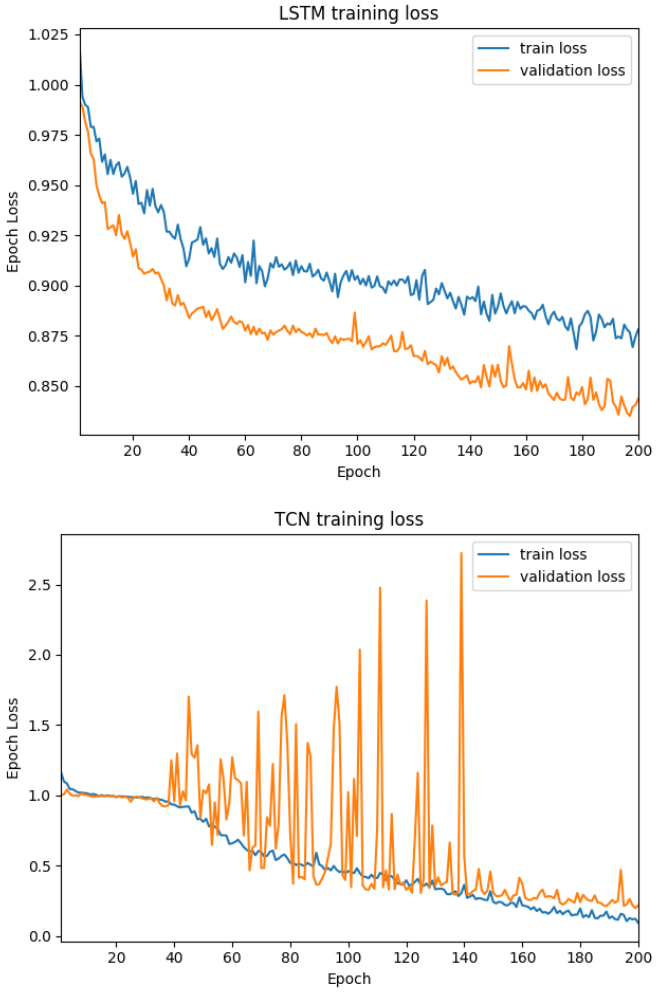

Fig. 7. The learning curves of the LSTM and TCN trained with the complete training set.

The most insightful results for the traditional machine learning algorithms come from the decision tree. This will shed light on what is considered to be important as predictors in the dataset. We also want to study how the tree changes as more data becomes available. We therefore look at the learned

structure of the decision tree trained with 400 examples (see Figure 8), and the one trained with 1600 examples (see Figure 9). While the first one has 6 levels, the last one has 8, showing that it learned more specific decision rules from the extra training data. We can also see that the feature *SE min* has a bigger importance that the others, as it is always the root node of the tree.

Fig. 8. Resulting decision tree using 400 examples.

We also plotted the confusion matrices of the decision tree after training it with 400 and 1600 examples. The extra examples help the decision tree to better distinguish between weak and severe leakage classes, although it already reached a very good precision with fewer training data (see figures 10 and 11).

VI. Conclusion

In this paper, we compared two deep learning models with other more traditional feature engineering-based machine learning algorithms in a real-life PdM use case. From the experiments, we were able to show that less complex algorithms can learn with fewer examples than deep learning, likely caused by the smaller parameter set (see, e.g., [14]). This result confirms that basic machine learning methods combined with a simple feature engineering technique can be a very good choice for PdM cases where the data is limited.

On top of that, we compared the data efficiency of the TCN, a new and unexplored deep learning topology in the PdM domain, with the popular LSTM. The results showed that the TCN can reach a good performance even with scarce data, while the LSTM was unable to learn from the same amount of data. This can be explained by the excess of parameters in the LSTM topology that come from all the fully connected layers of its modulating gates. The TCN, on the other hand, profits from the parameter sharing of the stacked dilated convolutions to efficiently learn the relationship between the sensors and the target output. While it did not outperform the feature engineering based algorithms, it did get very close with only fairly limited data, showing that they can be applicable in more PdM cases with relatively scarce data.

For future work, we want to study the behavior of the algorithms over more PdM datasets, also of slightly larger size to see whether the TCN can indeed outperform the feature-engineering based approaches and also LSTM architectures. Previous research has already shown that LSTMs can be the top performers, but they have not been compared to TCNs for those cases. In addition, we aim to study the performance of the algorithms in more complex environments with more complex relationships in the data and want to bring the best models in practice in a large industrial setting.

Acknowledgments

This work has been conducted as part of the Just in Time Maintenance project funded by the European Fund for Regional Development.

References

[1] A. K. Jardine, D. Lin, and D. Banjevic, "A review on machinery diagnostics and prognostics implementing condition-based maintenance," *Mechanical Systems and Signal Processing*, vol. 20, no. 7, pp. 1483 – 1510, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0888327005001512

[2] J. Lee, F. Wu, W. Zhao, M. Ghaffari, L. Liao, and D. Siegel, "Prognostics and health management design for rotary machinery systemsreviews, methodology and applications," *Mechanical Systems and Signal Processing*, vol. 42, no. 1, pp. 314 – 334, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0888327013002860

[3] O. Dragomir, R. Gouriveau, F. Dragomir, E. Minca, and N. Zerhouni, "Review of prognostic problem in condition-based maintenance," in *2009 European Control Conference (ECC) Proceedings*, 08 2009, pp. 1587–1592.

[4] X. Li, Q. Ding, and J.-Q. Sun, "Remaining useful life estimation in prognostics using deep convolution neural networks," *Reliability Engineering and System Safety*, vol. 172, pp. 1 – 11, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832017307779

[5] B. Zhang, S. Zhang, and W. Li, "Bearing performance degradation assessment using long short-term memory recurrent network," *Computers in Industry*, vol. 106, pp. 14 – 29, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166361518304640

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[7] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *CoRR*, vol. abs/1803.01271, 2018. [Online]. Available: http://arxiv.org/abs/1803.01271

[8] J. Zhang, P. Wang, R. Yan, and R. X. Gao, "Long short-term memory for machine remaining life prediction," *Journal of Manufacturing Systems*, vol. 48, pp. 78 – 86, 2018, special Issue on Smart Manufacturing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0278612518300803

[9] N. Helwig, E. Pignanelli, and A. Schtze, "Condition monitoring of a complex hydraulic system using multivariate statistics," in *2015 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings*, May 2015, pp. 210–215.

[10] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.

[11] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[12] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

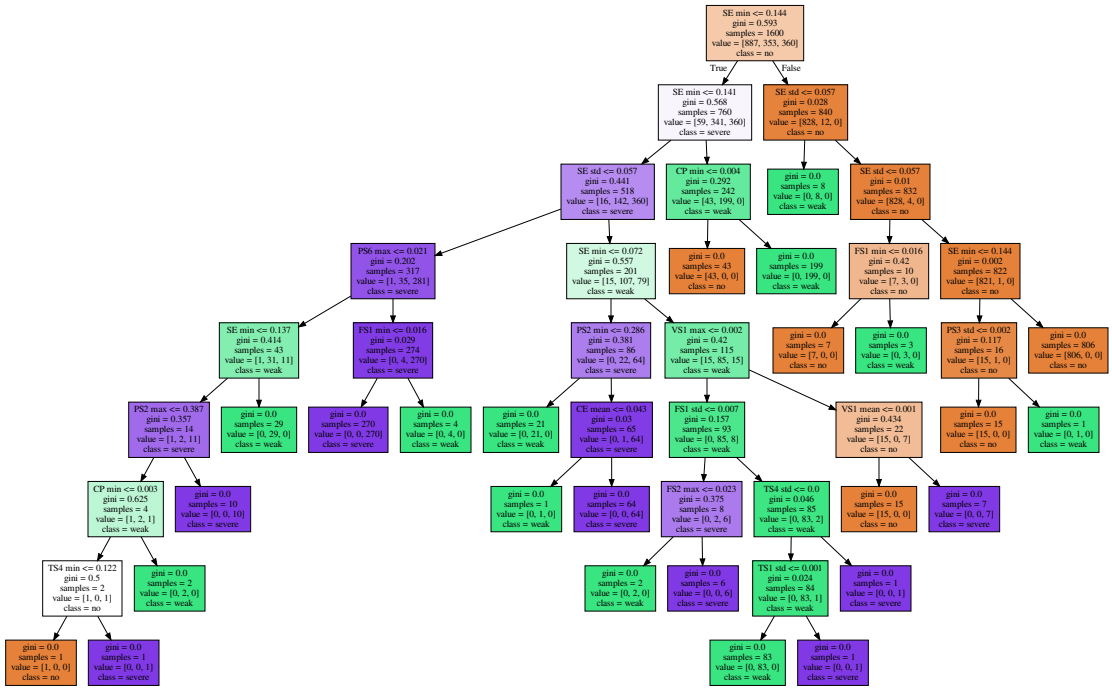

Fig. 9. Resulting decision tree using 1600 examples.

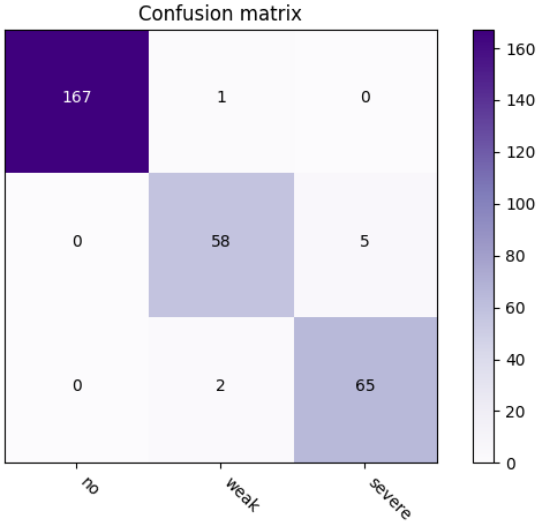

Fig. 10. The confusion matrix of the decision tree trained with 400 examples.

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[14] L. G. Valiant, "A theory of the learnable," in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*. ACM, 1984, pp. 436–445.

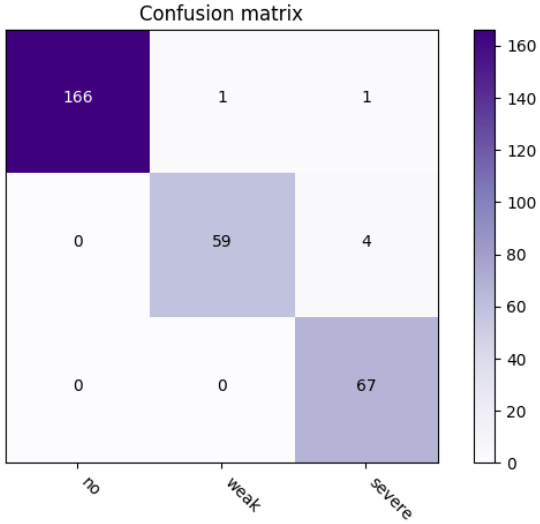

Fig. 11. The confusion matrix of the decision tree trained with 1600 examples.