Predicting machine failures from industrial time series data

Femke Jansen, Mike Holenderski, Tanir Ozcelebi
Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
Den Dolech 2, 5600 AZ Eindhoven, The Netherlands

Paulien Dam, Bas Tijsma
New Product Introduction
Philips Consumer Lifestyle
Oliemolenstraat 5, 9203 ZN Drachten, The Netherlands

***Abstract*—This paper addresses the problem of predicting machine failures in an industrial manufacturing process based on multivariate time series data. A workflow is presented for cleaning and preprocessing the data, and for training and evaluating a predictive model. Its implementation is modular and extensible to support changes in the underlying production processes and the gathered data. Two predictive models are presented, based on Convolutional Neural Networks and Recurrent Neural Networks, and evaluated on data from an advanced machining process used for cutting complex shapes into metal pieces.**

I. Introduction

The aim of smart factories is to exploit the data from various processes in the factory for optimizing these processes with the end goal of increasing customer satisfaction while reducing cost. It is then possible to use data driven methods for optimizing the maintenance of machines in a smart factory by predicting when machine failures will occur.

The leading example in this paper is a machining processes used for cutting complex shapes into metal pieces. The production line can be viewed as a chain of several processes and each process is implemented by several machines working in parallel. Each product travels through one machine in each process. During production, machine failures may occur. These failures can have different causes, such as an electrical malfunction (e.g. short circuit) or a mechanical malfunction (e.g. unable to move a tool). Moreover, all the tooling used in the production gets worn out in time, which can also lead to machine failures.

During the production data is gathered by the many sensors that are deployed around the production line, monitoring various process, material, and product quality parameters. Whenever a tool failure is detected by the monitoring system, it associates the product that was produced during the machine failure with an error code. The error code describes the cause of the failure and every cause has a unique error code. Upon a machine failure, a machine operator has to diagnose the failure and execute a corrective action.

The failure prediction problem can be expressed as a standard classification problem. The goal is to train a model using samples from past process data, classifying the samples as normal or likely to result in a failure (in a certain time interval). Then, during production, when a new measurement sample is collected, the model can predict whether the process is likely to fail or not. An accurate prediction has many advantages, such as the ability to schedule maintenance of multiple machines in a way that maximizes the number of products made per unit time and minimizes the cost of maintenance.

A typical challenge in training a machine learning model on industrial data is the class imbalance. Since the production processes have usually been already well designed and optimized, failures are rare, meaning that there are many more normal samples than failure samples. Consequently, it is difficult to train a model to identify failure samples.

Process samples are usually timestamped, and thus the data can usually be expressed as a multivariate time series, with each univariate time series representing a particular sensor. Common problems in industrial time series include irregular time intervals and missing values, due to misconfigured or failing sensors or data gathering infrastructure, or their maintenance and upgrade.

Finally, the data gathering was often designed for other purposes, for example for optimizing the product quality rather than predicting machine failures. Therefore, it is not always clear whether the information that is necessary for accurate failure prediction is present in the data and what prediction accuracy is achievable with the data.

This paper addresses the problem of training a classifier for predicting machine failures based on multivariate time series data with irregular timestamps and imbalanced classes. The proposed solution relies on established machine learning techniques for capturing relationships in highly dimensional temporal data, and a proposed generalization of a univariate time series analysis method to multivariate time series. The implementation is modular and extensible to support changes in the underlying production processes and consequently the gathered data.

II. Related work

This section provides an overview of common building blocks for a data processing pipeline.

A. Data cleaning and preprocessing

The configuration and data cleaning module usually forms the first step of the data preprocessing workflow. This

module is in charge of removing unwanted information, in particular missing values and irregular time intervals.

A common method for dealing with missing values is imputation, such as forward or backward filling [1]. Forward filling propagates the last non-missing value forward in time until another non-missing value is encountered, while backward filling propagates the last non-missing value backwards in time, until another non-missing value is encountered. Both approaches are simple, but they rely on the assumption that the value remains valid until another value is encountered.

Another method is interpolation [1], in which missing values are constructed based on nearby values. A simple method is linear interpolation. More complex methods employ polynomials of higher degree or splines. Although more complex interpolation methods may fit the data better and as such, may give better interpolation results, all of these methods rely on the assumption that a curve can be fitted which approximates the underlying distribution.

Instead of imputing missing values, rows containing them can be removed. By removing data, no assumptions are made with respect to the underlying distribution. This form of data cleaning can be done if there is no method available to reasonably impute the missing values (e.g. no underlying distribution is known), or if the number of missing values in the data is small.

In order to turn a time series with irregular time intervals into a periodic time series, so called resampling methods [2] are often used.

In the presence of imbalanced classes, it may be beneficial to reduce a multi-class classification problem to several binary classification problems [7]. For example, in the one-VS-all scheme, one class is treated as the positive class and the remaining classes are treated as the negative class. This process is repeated for every class, which reduces a multi-class problem to a binary classification problems. In one-VS-one, all pair-wise combinations of multiple classes are considered instead.

B. Sampling

Several well-known techniques can be used to create sets of samples for training and validation [3]. A common approach is validation (also known as hold-out), in which a single split of the data is used. A variant of this, k-fold cross validation, uses k random splits (or folds) of the data. Bootstrap involves randomly sampling data observations with replacement, which is typically used to increase the size of the training set. Unfortunately, none of these techniques can be used for time series data, because they do not take into account the temporal structure in the data.

Moving block bootstrap [4] is an alternative for bootstrap, that is compatible with time series data. The time series is broken into blocks of subsequent values, typically of equal size. The blocks need not be disjoint. Now, the blocks of time series are sampled with replacement to obtain one or more bootstrap sets. This form of bootstrap preserves the temporal structure within blocks, under the assumption that the structure between blocks is (almost) independently and identically distributed. The downside of this approach is that it cannot model long-term dependencies that span multiple blocks.

Two alternatives for k-fold cross validation are proposed in [5]. The first variant is called moving cross validation, where the idea is to use a fixed-length moving window to construct training and validation sets. The (sorted) time series data is first divided into k folds, of roughly equal size. Training and validation sets have $0 < k' < k$ folds, such that the first $k' - 1$ folds serve as the training data and the last fold serves as the validation data. The advantage of this approach is that it preserves the temporal structure of time series data within the individual folds and within $k' - 1$ (training) folds. However, choosing an appropriate k may be difficult.

In rolling cross validation, instead of moving a window over the sets, the sets are sequentially expanded. The time series data is again divided into k folds of roughly equal size, but now the training and validation sets are defined using a number of folds, $0 < k'' < k$. Since the sizes of the training sets keep increasing, it is possible to preserve temporal structure in the time series data up to $k-1$ folds. As with moving cross validation, a proper choice of k remains important however.

Rather than choosing the number of folds k and splitting the data into k parts of roughly equal size, it is also possible to specify the exact number of rows to define a time series sample. This means that we take into account a certain history length h. This approach is taken in [6].

When classes are unbalanced it can be difficult to determine what exactly characterizes a minority class sample. In random undersampling [8], the idea is to remove a randomly selected set of majority class samples from the original dataset. Random oversampling, on the other hand, duplicates data samples from the minority class to adjust the class distribution. However, duplicating existing samples might cause the classifier to overfit the data, so undersampling is usually preferred. Alternatively, SMOTE [9] is an oversampling technique, where a new data point is created by taking a random point along the vector between a randomly selected minority sample and one of its nearest neighbors. However, SMOTE performs poorly in high dimensional spaces and it is not clear how to interpolate categorical values (e.g. error codes).

C. Predictive modeling

Convolutional Neural Network (CNN) [14], [20] is a popular deep learning model for automatically extracting high level features from raw data which is suitable for solving classification or regression problems. Convolving filters over the input vector allows to reduce the number of trained parameters, thus reducing the resource requirements for training and storing CNN models. CNNs have regular feed-forward connections, making them suitable for parallelizing the training (e.g. using GPUs). The design of CNN architectures for solving a particular machine learning task, however, still remains an art.

For classifying univariate time series, an interesting approach is described in [13]. Here, each time series sample is converted to an image, using a technique known as

Gramian Angular Field (GAF). The basic idea is to encode the temporal correlations within each time series sample, yielding a heat map. To the best of our knowledge, GAF has been applied to univariate time series only. In this paper we extend the CNN+GAF procedure to multivariate time series.

Recurrent Neural Network (RNN) [17], [19] is a popular deep learning model for analyzing sequences. The recurrent connections effectively allow RNNs to share parameters and thus capture long time dependencies using simpler models than deep feed-forward neural networks, making them suitable for cases with smaller data sets than needed for some other deep networks. However, they are more difficult to parallelize (e.g. using GPUs) than feed-forward networks, due to the sequential dependencies between the states. RNNs, and LSTMs [18] can be effective for modeling long-term time dependencies in multivariate time series.

III. Data processing workflow

The proposed data processing workflow is illustrated in Figure 1. It focuses on the preprocessing steps for preparing the data for the training of the predictor (performed in the Forward Selection module). Each block in the workflow is a

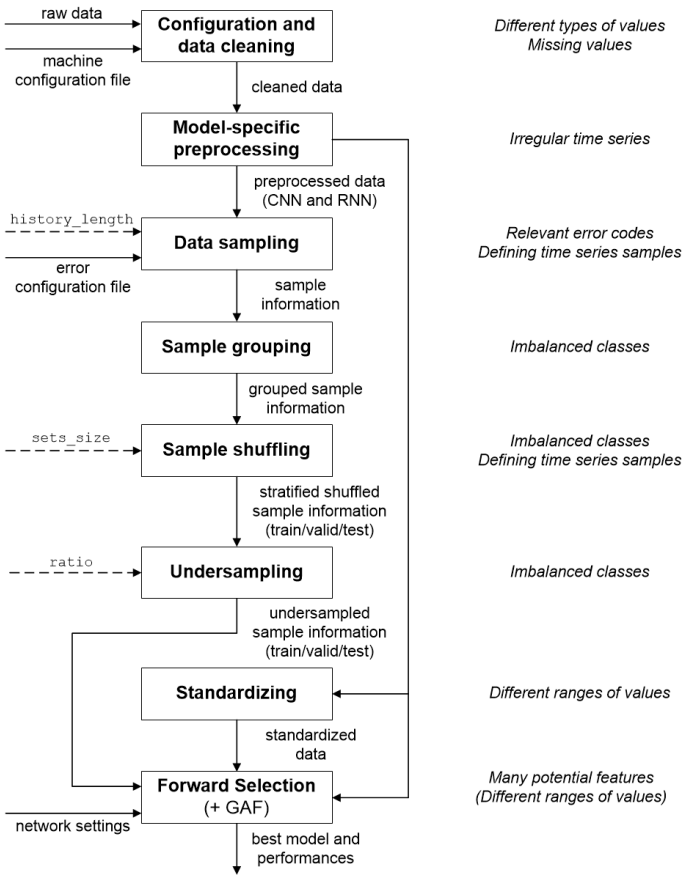

Fig. 1. Data processing workflow. Each block represents a module. Solid arrows indicate data flow and dashed arrows represent input parameters. The data characteristics that are handled by each module are also shown.

module that implements a group of preprocessing operations. The modules are executed in the given order. Each module produces one or more outputs for subsequent modules.

The workflow is implemented in Python, and relies on the pandas library. Extensibility of the workflow is provided by changing the parameters of the preprocessing modules, as well as by substituting concrete module implementations. The interfaces between the modules (i.e. the structure of the data frames), however, are fixed.

A. Configuration and data cleaning

This module is in charge of removing unwanted information, in particular missing values, from the data. The most common approach is to fill in (or impute) missing values, based on a certain scheme. The raw data is assumed to reside in a Comma Separated Value (CSV) file. The *machine configuration file* contains the metadata, describing the semantics of each column (e.g. its type) and which columns should be used in the later modules (to select only the relevant features). It simplifies adapting the workflow to other data sets and accommodate changes to the production process (e.g. new sensors being added or replaced).

B. Model-specific preprocessing

This module performs any other preprocessing on the data, on a per-model basis if necessary. The goal here is to further remove parts of the data (e.g. columns), to make the data compatible with specific models. If another model is added later on that requires different preprocessing, this module can be used to implement the desired preprocessing steps and produce a separate output file for the model in question. If no further preprocessing is required, this module is optional.

For the two neural network based predictive models considered in this work (CNN and RNN, see below), the timestamp column is removed from the data. A CNN does not need a timestamp column, because it uses image data as input (which does not contain a date time component). For an RNN, only the sequence of values matters and not the exact timestamp values. An ordered sequence of natural numbers is used instead of timestamps in subsequent modules.

C. Data sampling and sample grouping

These two modules combine consecutive rows to form samples with a history length $h \geq 1$, specified by the *history_length* parameter. The *error configuration file* specifies a list of error codes which should be included or excluded, to support the input of expert knowledge (e.g. process engineers may decide to exclude samples with irrelevant error codes).

D. Sample shuffling

This module divides the samples in training, validation and test sets, using a combination of ordinary validation [3] with moving block bootstrap [4]. To handle unbalanced classes, the sets are stratified, meaning that the original distribution of normal versus failure samples is maintained in each set. The *sets_sizes* parameter specifies the desired sizes of the three sets.

E. Undersampling

This module adjusts the class distributions of failure versus normal samples. The *ratio* parameter allows to set the desired ratio between the majority and minority classes.

F. Standardizing

This module performs data rescaling operations, to make sure that the ranges of values are appropriate for the models that are trained later on, e.g. to speed up their training. In this work all columns are standardized to 0 mean and standard deviation of 1, to avoid common problems with the training of the neural network models in the next module.

G. Forward selection

This module is responsible for training and selecting the best predictive model (a classifier). In the presence of many raw features (i.e. columns), selecting a simpler model with only a subset of the available features may result in better performance, both in terms of training speed and prediction accuracy. This module uses the forward selection method [11] for this purpose.

Two neural network based models are implemented, which can be configured with the *network_settings* parameter, which provides network model specific settings.

1) CNN + GAF: CNNs are inspired by the visual cortex in a human brain. The neurons in a typical CNN are arranged in layers, with the main building blocks being the convolution and polling layers, intertwined with standard layers, (e.g. dense or softmax). Each neuron in a convolutional layer, connects to a small part of the input layer (e.g. 3 by 3) referred to as its local receptive field. All the weights and biases of the neurons in the convolutional layer are shared. This allows the network to detect a particular feature in the input image at any position. The objective of pooling is to simplify or summarize the information from a convolutional layer in a small square region for a particular feature map. The size of the pooling region is also known as the pooling size. In this work max-pooling os epmloyed, where a pooling unit outputs the maximum activation in its pooling region. Pooling reduces the number of parameters of a CNN, because it reduces the size of the remaining input image (after applying convolution).

For a CNN, we use the GAF procedure [13] to transform each dimension in a multivariate time series into a single channel in an image, resulting in a multi-channel image. We call this generalization Multi-GAF. CNNs can work with multi channel images generated by Multi-GAF by using fully connected layers at the end. The GAF procedure is intertwined with the forward selection module, because we need to construct many different multi-channel images, depending on the variables that are selected. The GAF procedure also makes sure that the resulting values are in an appropriate range for CNNs. Figure 2 illustrates the result of the GAF procedure applied to several individual features of a sample.

The CNN architecture used in this work consists of an input layer (the Multi-GAF transformation applied to the output of the Standardizing module), one or more convolution-convolution-pooling layers, two dense layers and a softmax layer.

2) RNN: RNNs are used for processing temporal sequences (also of variable length), through recurrent connections, which allow the network to maintain a state, "remembering" previously ingested samples. While the general RNNs are difficult to train using the common gradient descent methods (due to vanishing or exploding gradients), the Long Short Term Memory (LSTM) architecture of RNNs allow to effectively capture long-term dependencies. It introduces the concept of gated memory cells, which control when information is stored in a cell, when it is propagated further into the network and when it is forgotten. Gates are implemented using standard neural network primitives (e.g. weight parameters, dot products, linear and non-linear functions) and can be trained using the standard stochastic gradient descent optimization.

After one LSTM layer, one or more LSTM layers can be added to improve classification performance [15]. Each subsequent LSTM layer receives input from the previous LSTM layer and itself. The RNN network used in this work consists of an input layer (the output of the Standardizing module), one or more LSTM layers, and one softmax layer.

IV. Evaluation

In this section we instantiate the presented data processing workflow for predicting machine failures in the industrial use case introduced in Section I.

A. Data

The data is in the form of a table, and contains measurements of the process parameters of one machine in the production process, with rows representing measurement samples and columns representing the different sensors. One column describes the state of the machine, i.e. if it is in a normal state or a failure state (in which case it contains the error code). This column represents the labels for the rows. Failure rows contain random process parameter measurements, which are often outside the specification window. The original data set contains around 260000 rows, with only 900 failure rows (with 20 possible error codes) and 21 columns.

1) Configuration and data cleaning: There are actually only few missing values in the data (less than 0.03%), so it was decided to remove rows containing missing values. Moreover, we use the one-VS-all scheme, and merge all the error codes into one, grouping the samples as either normal or failure, to reduce the class imbalance.

B. Data sampling and sample grouping

In this work, samples are defined using a fixed number of rows (the history length h), as done in [6]. Each sample is labeled as *normal* or *failure*. Since the process parameters in a failure row are typically erroneous, we choose to ignore those parameters and for each label in row i we create a sample using process parameters from rows $i-1$ to $i-1-h$. If we have n rows in the data and a history length of h, we can thus define at most $n-h-1$ samples this way.

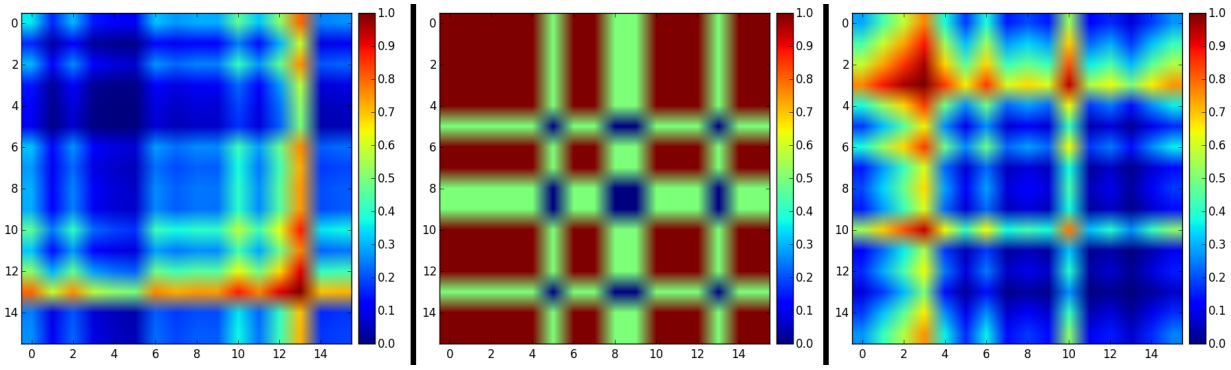

Fig. 2. Illustration of the individual (1-channel) images produced by the Multi-GAF procedure, for three process columns. The numbers near the X- and Y-axis indicate the history length for the sample (in this case 16). The colors indicate the strength of the correlation between two values of the time series sample, where 1/red is a strong correlation and 0/blue is a weak correlation.

To prevent overlapping samples with inconsistent labels, all normal samples that would overlap with failure samples are removed from the set of samples. Normal samples are removed through undersampling, instead of failure samples, because of the high class imbalance with limited number of failure rows.

C. Undersampling

Undersampling ratio 4 is used.

D. Modeling

We have evaluated the CNN and RNN based methods on a data set that in the end was reduced to 450 failure and 1800 normal samples. The best performing models returned by the forward selection procedure relied on 3 features. We have investigated different history lengths (16, 32, 64 rows) and different network sizes (1, 2, 3 convolution-convolution-pooling or LSTM layers described in Section III-G1). The CNNs were using 3x3 convolutional filters, 2x2 max poolings, stride of 1, 32 feature maps per convolutional layer and two dense layers. The RNNs were using standard LSTM layers of width 64, 128, or 256.

To avoid overfitting, dropout [16] with 0.5 drop probability was used in the first dense layer in CNN and in each LSTM layer in RNN. No dropout is applied in other layers of the CNN, as the convolution and pooling layers naturally impose some regularization by restricting the neurons in these layers to connecting to a small region of the input and by sharing weights.

Both models were implemented using the Keras [12] library for Python and trained with early stopping with patience of 10 samples.

E. Results

The results for history length 64 are summarized in Figures 3 and 4, showing that the proposed RNN method outperforms the proposed CNN method. Training a single CNN and RNN took on average around 1.7h and 2.7h, respectively, on a PC with 8GB of memory and 2.1GHz Intel Core i7 processor and NVIDIA GeForce GT 720M with 2GB memory.

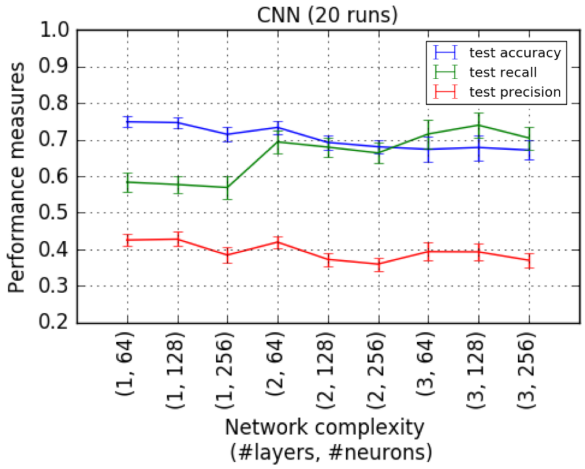

Fig. 3. Evaluation results for the CNN+GAF model.

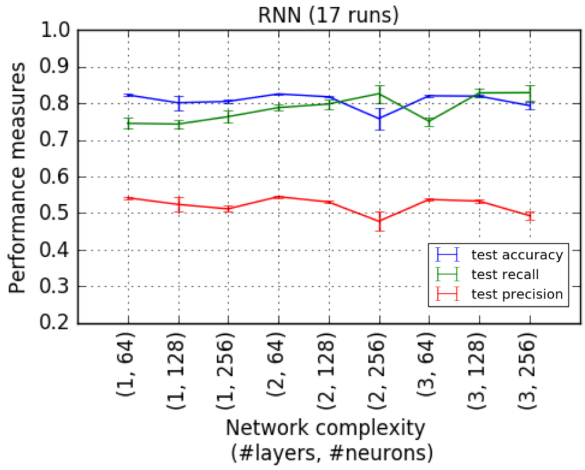

Fig. 4. Evaluation results for the RNN model.

Since there is no baseline machine failure predicting model at the manufacturer to compare against, it is difficult to judge how well the proposed solutions are performing. To put the results in perspective, however, we can compare them to a trivial classifier whose predictions are failure with probability $P = 1$. Let F be the probability of a true

failure sample (in our example it is equal to the fraction of failure samples, i.e. $F = 0.2$). The probability that the trivial classifier will return a true positive is $P * F$. Hence, we can compute the expected recall and precision of the trivial classifier to be:

$$\begin{array}{lcl} \text{Recall} & = & \frac{P*F}{F} = P = 1 \\ \text{Precision} & = & \frac{P*F}{P} = F = 0.2 \\ \text{Accuracy} & = & P * F = 0.2 \end{array}$$

The decision whether the proposed CNN and RNN methods provide a good recall and precision trade-off is for the process engineers. However, computing the F1 scores can give us an indication whether we have learned anything compared to the trivial classifier. If we pick the (2, 64) RNN configuration, then

$$\begin{array}{lcl} F1_{RNN} & = & \frac{2*0.55*0.83}{0.55+0.83} = 0.66 \\ F1_{trivial} & = & \frac{2*1*0.2}{1+0.2} = 0.33 \end{array}$$

which shows that the RNN performs better than the trivial classifier.

Since the optimal recall, precision and F1 score values are 1, the results show that there is still room for improvement. One possible explanation for the performance is overfitting. After the data preprocessing, the models were trained on 2350 samples (450 failure and 1800 normal) with 3 features. While both CNN and RNN models employed dropout for regularization, their large number of parameters at some point may have saturated. Figures 3 and 4 show that the CNN model performance decreases slightly with increasing model complexity, while the performance of the RNN model remains approximately constant. This suggests that indeed, in case of CNN, the model may be overfitting.

However, since the data was gathered for other purposes (which is typical for industrial use cases that tend to reuse the available data for different purposes), it is not clear whether the data actually contains enough information to do better, and what the optimum performance is on this data on this task.

V. Conclusion

A modular data driven workflow was presented for predicting machine failures in an industrial process. Two neural network based predictive models were evaluated, one based on a generalization of the univariate CNN+GAF combination applied to multivariate time series. The second one was based on standard LSTMs. The two models were evaluated on data from a machining process used for cutting complex shapes into metal pieces. The results show that both models could learn to predict some of the failures, with the LSTM model performing better. There is room for improvement, but it is not clear what the optimum performance is for predicting machine failures using this data.

As future work, alternative methods for failure prediction can be investigated (e.g. based on other computational models such as spiking neural networks which seem to be resilient to missing data), as well as methods for estimating the information content in the data with respect to solving a particular machine learning task. Also, alternative sampling methods can be investigated, where the good samples are taken just after a machine was maintained, rather than just before a normal state.

Acknowledgment

This work has been partly supported by the EU ECSEL JU grant nr. 662189 (MANTIS).

References

[1] G. James, D. Witten, T. Hastie, R. Tibshirani, *An Introduction to Statistical Learning*, Springer, 2014.

[2] K. Rehfeld, N. Marwan, J. Heitzig, J. Kurths, *Comparison of correlation analysis techniques for irregularly sampled time series*, Nonlinear Processes in Geophysics, 18:389 – 404, 2011.

[3] S. Arlot, A. Celisse, *A survey of cross-validation procedures for model selection*, Statistics Surveys, 4:40 – 79, 2010.

[4] J. Berkowitz, L. Kilian, *Recent developments in bootstrapping time series*, 19:1–48, Econometric Reviews, 2000

[5] M. Y. Hu, G. Zhang, C. X. Jiang, B. E. Patuwo, *A cross- validation analysis of neural network out-of-sample performance in exchange rate forecasting*. Decision Sciences, 30(1):197 – 216, 1999.

[6] J. F. Murray, G F. Hughes, K. Kreutz-Delgado. *Machine learning methods for predicting failures in hard drives: A multiple-instance application*, Journal of Machine Learning research, 6:783 – 816, 2005.

[7] C. M. Bischop *Pattern Recognition and Machine Learning*, Springer, 2006.

[8] H. He, E. A. Garcia, *Learning from imbalanced data*, IEEE Transactions on Knowledge and Data Engineering, 21(9):1263 – 1284, 2009.

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, *SMOTE: Synthetic minority over-sampling technique*, Journal of Artificial Intelligence Research, 16:321 – 357, 2002.

[10] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista, *The UCR time series classification archive*, http://www.cs.ucr.edu/∼eamonn/time_series_data, 2015

[11] C. Chen, D. B. Kaber, P. G. Dempsey, *Using feedforward neural networks and forward selection of input variables for an ergonomics data classification problem*, Human Factors and Ergonomics in Manufacturing & Service Industries, 14(1):31 – 49, 2004.

[12] Keras documentation. https://keras.io

[13] Z. Wang, T. Oates, *Spatially encoding temporal correlations to classify temporal data using convolutional neural networks*, CoRR, 2015.

[14] Y. LeCun, Y. Bengio, *Convolutional networks for images, speech, and time series*, The handbook of brain theory and neural networks, MIT Press, 1998

[15] X. Li, X. Wu, *Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition*, IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. *Dropout: A simple way to prevent neural networks from overfitting*, Journal of Machine Learning Research, 15:1929 – 1958, 2014.

[17] J. B. Zachary, C. Lipton, C. Elkan, *A critical review of recurrent neural networks for sequence learning*, CoRR, 2015.

[18] F. A. Gers, J. Schmidhuber, F. Cummins, *Learning to forget: continual prediction with LSTM*, Neural Computation, 12(10):2451 – 2471, 2000.

[19] E. Choi, A. Schuetz, W. F. Stewart, J. Sun, *Using recurrent neural network models for early detection of heart failure onset*, Journal of the American Medical Informatics Association, Oxford University Press, 2017

[20] Y. Lin, C. Sakr, Y. Kim, N. Shanbhag, *PredictiveNet: An energy-efficient convolutional neural network via zero prediction*, IEEE International Symposium on Circuits and Systems (ISCAS), 2017