CS341 Project Report
Data-driven Predictive Maintenance

Peng Yuan
pengy@stanford.edu

Rao Zhang
zhangrao@stanford.edu

Pan Hu
panhu@stanford.edu

Abstract

Modern machines usually have many sensors monitoring and logging its working status, which provides a large amount of data, making data-driven predictive maintenance possible. Traditional prediction methods involve feature engineering and usually use a SVM or random forest to predict. Recent years, deep neural networks demonstrate their power in many tasks, and we believe they can be very useful in the predictive maintenance task as well. In this work, we propose a CNN+LSTM model with a novel shared normalization method, which significantly improved the prediction performance over the SVM model (from 0.57 AUC to 0.7549 AUC). Our extensive experiments also show that data balancing, normalization, and regularization are several key factors in the task as well. The source code and our best model is stored in our Github repository[1].

1. Introduction

Intelligent heating systems nowadays are equipped with various powerful sensors to monitor operating status, including power consumption, temperature, flow rate, switch status, etc. Predicting when a heating system will fail from various sensor readings and scheduling maintenance ahead of time could potentially save millions of dollars in repair.

Problem definition. We define our problem as: *given all sensor data of one heater, for each day, predict whether it will fail in the next 7 days*. More specifically, assuming that the sensor fails on Monday of week 2, as shown in Figure. 1, we examine each day in week 1 to predict whether it will fail in the next 7 days.

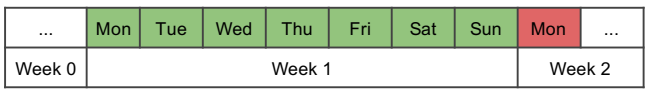

Figure 1: Schedule maintenance before system fails.

In this paper, we proposed a deep learning method to tackle the prediction problem, which combines Convolutional Neural Network (CNN) and Long-Short Term Memory Network (LSTM) [1]. In addition, we introduced a shared normalization method which can better preserve the correlation information in the original data, and thus further improve the overall performance.

The paper is organized as follows. We first introduce related works that are trying to solve similar problems with us, then describe details about the dataset include statistics of the data, challenges in prediction as well as how we prepare the data for classification/prediction. After that we introduce traditional and deep learning methods to tackle the problem, followed by experiments and benchmarks comparing the performance of each method. At the end we conclude our paper and propose future works that could potentially improve prediction performance.

2. Related Works

Time series prediction/classification. There are extensive works on time series prediction and classification. The most widely used statistical methods include: autoregressive moving average model (ARMA) [2] and autoregressive integrated moving average (ARIMA) [3, 4]. However, models in these methods tend to include only a few parameters, thus limit their expression capability, especially when the dataset is very large. The same analysis also applies to regression, SVM or random forest especially with small amount of trees

Deep learning methods. Recent deep learning architectures, especially RNN/LSTM/CNN have gain its popularity in prediction/classification tasks on time series data [5, 6, 7, 8]. In 2016, Zhang. et al proposed Deep Symbolic Representation Learning [9] which also targeted at data-driven prediction for intelligent heating systems. The key difference between Deep Symbolic Representation Learning and our work is that, we propose a different deep learning network architecture by combining LSTM and CNN, while the solution proposed in Deep Symbolic Representation Learning is closer to vanilla LSTM.

3. The BOSCH Dataset

Our dataset is a private dataset provided by BOSCH that requires signing non-disclosure agreement (NDA). Hence we can only show some basic information of the dataset (Table 1). The distribution on number of days per system is shown in Figure 2.

[1] https://github.com/frankpengyuan/predictive-maintenance

Table 1: BOSCH dataset

Sensors per heater	106
Number of heaters	508
Total failures	307
Size of data	277 GB
Time span	2014-01-03 - 2015-12-08

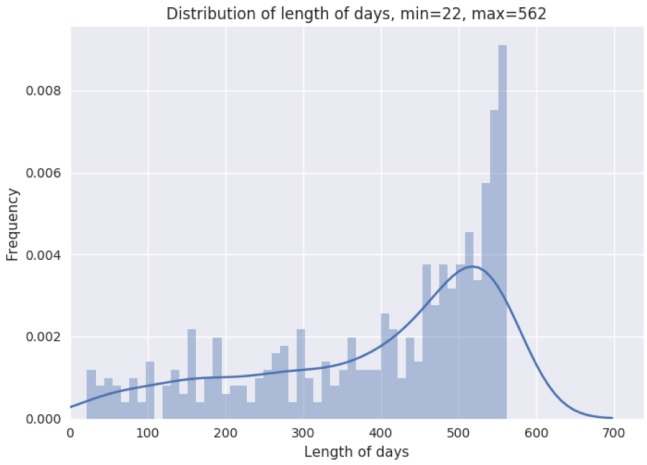

Figure 2: Distribution on number of days per system. Min=22, Max=562

The original data is organized in an event-driven manner: at a time stamp, one or more sensors report some readings. It is also possible that one sensor report multiple readings at the same time. In this case, we drops all the duplicate records, except the last one, which contains all the information prior to this time stamp.

3.1. Labeling

We label the data as follows: start where a failure occurs, label 7 days before it as *positive days*, and then drop the exact day on which the failure happens, and label all the rest days as *negative days*. If there isn't any failure in a heater's whole records, then label them all as *negative days*. If the record misses some days that should be labeled as positive days, just ignore them.

3.2. Challenges

Scale of the dataset. The data on average spans 300 days $\times$ 24 hr/d $\times$ 60 min/hr $\times$ 30 records/minute. Given that each records consists of 106 sensors readings and there are about 500 such systems, there are nearly 8 billion 106-dimensional records in total.

Noise. Sensor logs are always noisy, and what even worse in our case is that special actions like rebooting etc. will result in unexpected data, like the temperature may be -3276.8 Fahrenheit degree in some cases. In addition, fluctuations in environment (eg. temperature) also contribute to the noise, making it hard to distinguish real changes in data from changes caused by noise (which influence some extracted features later).

Imbalance of labels. The dataset is extremely unbalanced. Failure occurs only once in a while. Most heaters do not fail at all during the whole time period. There are only 2,174 positive days (after labeling), and however, more than 100,000 negative days in the dataset.

3.3. Data preparation

Focus on key sensors. Our predictions are based on 19 major (out of 106) sensors shown in Table 2. They are selected by domain experts and more likely to reflect system status changes.

Table 2: Focused Sensors

No.	Sensor
1.	Actual_Power
2.	Number_of_burner_starts
3.	Operating_status:_Central_heating_active
4.	Operating_status:_Hot_water_active
5.	Operating_status:_Flame
6.	Relay_status:_Gasvalve
7.	Relay_status:_Fan
8.	Relay_status:_Ignition
9.	Relay_status:_CH_pump
10.	Relay_status:_internal_3-way-valve
11.	Relay_status:_HW_circulation_pump
12.	Supply_temperature_(primary_flow _temperature)_setpoint
13.	Supply_temperature_(primary_flow _temperature)
14.	CH_pump_modulation
15.	Maximum_supply_(primary_flow) _temperature
16.	Hot_water_temperature_setpoint
17.	Hot_water_outlet_temperature
18.	Actual_flow_rate_turbine
19.	Fan_speed

Cleaning. We remove all almost-empty files that don't contain enough information for analysis, leaving 484 heaters for training and testing. We also remove contaminated records appearing at the very beginning of each file and shortly just after each system reboot.

Resampling. The data is then resampled by day, with an interval of 10 seconds. Hence, every day's data is aligned to the same length, say 8640 records. This step is helpful for feeding data into predicting systems with fixed input size.

4. Models

In this section we introduce traditional and deep learning modeling to tackle the prediction problem.

4.1. Traditional models

We reframe our prediction task as a binary classification problem on numerical time-series data. Hence, we try traditional classification methods first and use them as baseline later. Among all traditional methods, we selectively pick linear SVM and RBF kernel SVM as our models because **1)** linear SVM could represent all linear classification methods family and it is fast to train and test; and **2)** RBF kernel SVM usually gives better classification results, though it is slower to train.

4.1.1 Linear SVM

Linear SVM was first proposed by Vapnik in 1963 [10], which aims to binarily separate high-dimensional data points (or feature representation) with a hyperplane. A set of hyperplane in the input space can be defined as $w \cdot x - b = 0$. Then we can select two parallel hyperplane in the set to separate the data points, which can be described in Equation 1.

$$\begin{aligned} w \cdot x - b &= 1 \\ w \cdot x - b &= -1 \end{aligned} \tag{1}$$

The region bounded by these two hyperplanes is called the "margin". There are two types of linear SVM according to the margin and separability. For data points linearly separable, hard-margin SVM is usually applied, while for data points not linearly separable, soft-margin SVM is usually adopted.

4.1.2 RBF kernel SVM

Non-linear SVM classifier has a very similar form as the linear ones, which replaces the dot product by some non-linear kernel functions. It is therefore able to solve the maximum-margin optimization problem in transformed input space. This idea was first introduced by Aizerman *et al.* in [11].

The (Gaussian) Radial Basis Function (RBF) is one famous non-linear kernel function in various learning algorithms. Especially it is widely used in Support Vector Machine (SVM) classification.

Here, we denote x and x' as two samples, representing features in the input space. Then RBF can be defined as follows in Equation 2

$$K(x, x') = e^{(-\frac{||x-x'||^2}{2\sigma^2})} \tag{2}$$

where σ is a hyperparameter and $||x - x'||^2$ is regarded as squared Euclidean distance.

4.2. Deep learning models

Deep neural networks have shown great capabilities in classification problems, and thus we adopted them in our problem as well.

4.2.1 Vanilla LSTM

LSTM is a very popular model for time-series data analysis since it can capture both the temporal information and the long-term ones. The original model is described in paper [1] and applying it on our dataset leads to a huge improvement (details are in the experiment section). As shown in Figure 3, we treat each day's data separately and feed it into the LSTM model, and apply a fully connected layer on the final output (hidden state) of LSTM to get the score for each class, and use a softmax for classification.

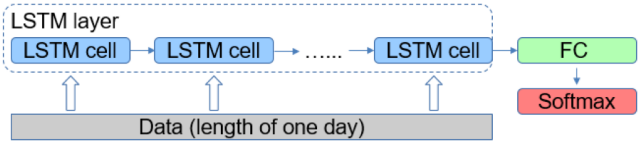

Figure 3: LSTM model architecture

4.2.2 CNN+LSTM

Convolutional neural network (CNN) is usually used in computer vision tasks and the convolutional layer can capture the localization information [12]. Nevertheless, a paper [13] suggests that a combination of CNN and LSTM can yield promising results on COCO2014 dataset [14]. In our case, we believe that the signal of failing heaters have similar local patterns along the time axis, thus apply a 1-dimension convolution layer could help find such patterns and help the LSTM to perform better. Therefore, we propose several CNN+LSTM models, and their architectures are shown in Figure 4.

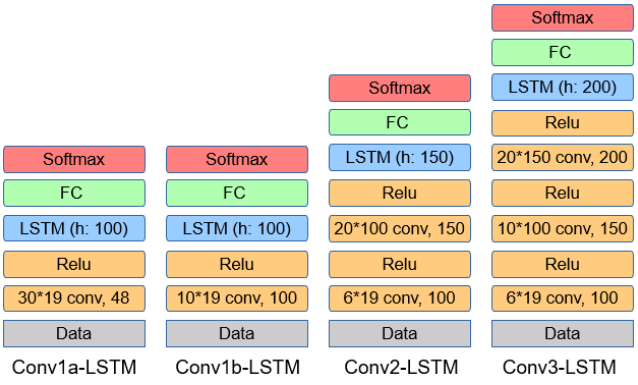

Figure 4: CNN+LSTM architectures (h is the number of hidden units in LSTM. "L*D conv, N" in the figure means N 1-dimensional convolution filters of size L, convolute on D input channels and output a new N features. All convolutional layers have stride=1 and use "valid" padding)

We set a stride equal to 1 in all convolutional layers since we have a LSTM after the convolutional part, which handles what information to keep or drop, and thus we don't have pooling layers and set the stride to 1.

Conv1a-LSTM and Conv1b-LSTM only contains 1 convolutional layer, but they have different number of filters and different filter size. Each filter in Conv1a-LSTM covers a 5-minute window of the data (as we re-sampled the data every 10 second) and each filter in Conv1b-LSTM covers a 100-second window.

Conv2-LSTM and Conv3-LSTM have multiple convolutional layers, and their receptive field is roughly 5 minutes as well, since we are using stride equals to 1 in all convolutional layers.

5. Experiments

We have in total 484 valid heaters, and we split it into 400 heaters for training and 84 heaters for testing (**large** set). In addition, we picked two subsets to speed up model selection and hyper-parameter tuning. The **small** subset contains 60 heaters in total, and they are *manually picked from heaters with top positive days*, and then randomly split into 2 groups: 50 heaters for training and 10 heaters for testing. Notice that the manually picked small set may have generalization issue, as we encountered in the later experiments. The **medium** subset randomly selects 200 heaters for training and 50 heaters for testing. As for the metric, we use Area Under ROC Curve (AUC) to measure the performance of our models.

5.1. Traditional models

The linear SVM is implemented with TensorFlow and run on a Tesla K80 GPU, while the RBF kernel SVM is based on scikit-learn library and run on CPU. We use a weighted loss to compensate the imbalance of the data.

Feature extraction. The SVM models usually require human-engineered features to work well, and we encoded each day's data into a large feature vector. More precisely, we split each day's data is split into 24 (or 48, 72) segments, and then use the minimum, maximum, standard deviation, first record, last record and frequency of changing within each segment as our features. In addition, we also add features which are the running average of above features to track the normal state for each sensor.

Table 3: SVMs test results

Model	C	gamma	loss weights	number of segments	AUC
Linear SVM	1	-	1:10	24	0.50
Linear SVM	0.1	-	1:10	24	0.50
Linear SVM	0.1	-	1:5	24	0.47
Linear SVM	0.1	-	1:10	48	**0.51**
Linear SVM	0.1	-	1:10	72	0.51
RBF kernel SVM	0.1	1e-7	1:10	24	0.46
RBF kernel SVM	0.1	1e-7	1:10	48	0.51
RBF kernel SVM	0.1	1e-7	1:10	72	0.50
RBF kernel SVM	0.01	1e-7	1:10	48	**0.57**
RBF kernel SVM	0.1	1e-9	1:10	48	0.52

Results. The results are shown in Table 3, and linear SVM can hardly separate the data into two classes while the RBF kernel SVM performs just slightly better than random. The failure of traditional methods on our problem once again shows the difficulty of itself. We believe that losing temporal information during feature extraction contributes most to the failure. Though extracting different features may improve the performance, it can hardly be generalized to other predictive maintenance problems and thus we switch to deep learning approaches and leverage the power of data.

5.2. Deep neural network models

We implement our deep neural network models with TensorFlow and train and test them with Tesla K80 GPU on Google Cloud. Training typically takes 6-8 hours per 1000 steps, depends on the model complexity. Each day's data is treated separately in the model, meaning we don't pass any information from previous day to the next day. In addition, we use Adam optimizer and a learning rate of 0.001 for all models in this section.

Balance the data. We balance the data during training to avoid constantly predicting 0. The balance is done by first checking how many positive days in a given heater, and then randomly selecting negative days from the same heater according to the balance ratio; If there is too few positive days, we are selecting negative days so that the minimal number of days per heater (min_n in Table 4) is reached. There's no balance action during testing.

Normalization. We tried several ways of normalization, and all of them normalize the data column by column within each heater. In Table 4, c stands for the data of one column, and e.g. $c - ave(c)$ means for each column, the normalized data equals to the original data minus the average of that column. In addition, we also introduced **shared normalization** to take the column-wise correlation into account. More concretely, we combine column 12 and 13, 16 and 17, which are all measurements of temperature, and calculate their maximum/average, which will be used during normalization instead of using the maximum/average of only one column.

Dropout and regularization. To avoid over-fitting, we add dropout and regularization to our models. In all the models, the dropout is added only to the LSTM layer, and the regularization is added to the convolutional layers before the LSTM layer. No dropout/regularization is added to the final fully connected layer. We are using L2 regularization, and the regularization strength is the weight of the regularization loss (and the original loss has weight of 1.0).

Results. The results are shown in Table 4. We make the following observations from it:

1) Even the basic LSTM model improve the AUC significantly compared to the SVM models.

2) Normalization is critical. Without normalization, the models can hardly learn anything. We also notice that using *shared normalization* will always give a better result.

Table 4: Deep neural network models results. (batch size is 90, and model architectures and parameters are in Figure 4)

Dataset	Model	Balance (Neg:Pos)	min_n	Normalization	Dropout rate	Regularization strength	Iteration	Train AUC	Test AUC
Small	LSTM	2:1	15	None	0.0	-	3000	-	0.5214
		1:1	15	None	0.0	-	3000	-	0.5072
		2:1	15	$(c - ave(c))/max(c)$	0.0	-	3000	-	0.6937
		2:1	15	$(c - ave(c))/max(c)$	0.5	-	3000	-	**0.7102**
	Conv1a-LSTM	1:1	15	$(c - ave(c))/max(c)$	0.5	0.0	3000	0.7374	0.7460
		2:1	15	$(c - ave(c))/max(c)$	0.5	0.0	3000	0.7437	0.7556
		2:1	15	shared $(c - ave(c))/max(c)$	0.5	0.0	3000	**0.7581**	**0.7688**
	Conv1b-LSTM	2:1	15	shared $(c - ave(c))/max(c)$	0.5	0.0	3000	**0.7517**	**0.7701**
	Conv2-LSTM	2:1	15	$(c - ave(c))/max(c)$	0.0	0.0	3000	0.8722	0.6579
		2:1	15	$(c - ave(c))/max(c)$	0.0	0.0	2400	0.8618	0.6803
		2:1	15	$(c - ave(c))/max(c)$	0.5	0.0	4000	**0.8771**	0.5835
		2:1	15	$(c - ave(c))/max(c)$	0.5	0.0	3200	0.8574	0.6437
		2:1	15	$(c - ave(c))/max(c)$	0.0	0.01	2200	0.7212	0.7443
		2:1	15	$(c - ave(c))/std(c)$	0.0	0.01	2200	0.5150	0.5205
		2:1	15	$(c - ave(c))/5 \times std(c)$	0.0	0.01	2200	0.7118	0.7520
		2:1	15	shared $(c - ave(c))/max(c)$	0.0	0.01	2200	0.7202	**0.7555**
		2:1	15	shared $(c - ave(c))/max(c)$	0.0	0.001	2200	0.8402	0.7130
	Conv3-LSTM	2:1	15	shared $(c - ave(c))/max(c)$	0.0	0.0	2400	**0.7734**	0.7202
		2:1	15	shared $(c - ave(c))/max(c)$	0.0	0.01	2400	0.7374	**0.7435**
		3:1	15	shared $(c - ave(c))/max(c)$	0.0	0.01	2400	0.7200	0.7224
Medium	LSTM	2:1	15	shared $(c - ave(c))/max(c)$	0.5	-	3000	0.6977	0.6718
	Conv1a-LSTM				0.5	0.0	3000	0.7391	0.6946
	Conv1b-LSTM				0.5	0.0	1400	0.4526	0.4451
	Conv1b-LSTM				0.5	0.0	600	0.6911	0.6652
	Conv2-LSTM				0.0	0.01	1000	0.7526	0.7185
	Conv3-LSTM				0.0	0.01	1000	0.7403	0.7102
	Conv2-LSTM	2:1	10	shared $(c - ave(c))/max(c)$	0.0	0.01	1000	**0.7612**	**0.7233**
Large	Conv2-LSTM	2:1	10	shared $(c - ave(c))/max(c)$	0.0	0.01	2200	**0.7946**	0.7483
					0.0	0.01	1600	0.7871	0.7501
					0.0	0.01	1000	0.7818	**0.7549**

3) As the model grows deeper, dropout on LSTM layer can not prevent over-fitting effectively, and adding additional L2 regularization to the convolutional layers is necessary.

4) Conv3-LSTM performs worse than Conv2-LSTM, suggesting that adding too many convolutional layers may not be helpful.

5) Adding more data is helpful, as it boosts Conv2-LSTM by more than 3 percent on test AUC. However as we will use at least min_n days per heater, adding more heaters with no positive days will lead to imbalance of the data, and may reduce the performance of certain models like Conv1b-LSTM.

6) Conv1b-LSTM has a smaller receptive field, which helps it learn better over balanced data, but it may not be able to generalize well.

6. Conclusion

In conclusion, we explored several models in predictive maintenance task, and our Conv2-LSTM model achieved 0.7549 AUC on the test set (significantly improved over the SVM baseline which has 0.57 AUC), suggesting that deep neural networks is a powerful tool in predictive maintenance tasks. Our experiments also showed that using a shared normalization is helpful in data with column-wise correlation; and a combination of CNN and LSTM is indeed a good model for classifying numerical time-series data, but the depth of convolutional layers should be selected carefully.

In the future, several improvements are possible: **1)** Training and classifying on the entire 109 sensors instead of the 19 expert-selected ones. **2)** Carry each day's ending state in the LSTM to the next day, so that we have information passing through a series of days. **3)** We are currently label 7 days ahead as positive days; experiments on labeling 3 or 4 days ahead may provide more insights of the failure pattern.

7. Acknowledgement

We sincerely thank Rok Sosič and David Hallac for their insightful discussion and valuable suggestions from data processing to network architecture and training. We also appreciate BOSCH for providing the dataset and Google Cloud for providing the compute engine.

References

[1] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[2] S. E. Said and D. A. Dickey, "Testing for unit roots in autoregressive-moving average models of unknown order," *Biometrika*, vol. 71, no. 3, pp. 599–607, 1984.

[3] G. E. Box and D. A. Pierce, "Distribution of residual autocorrelations in autoregressive-integrated moving average time series models," *Journal of the American statistical Association*, vol. 65, no. 332, pp. 1509–1526, 1970.

[4] J. Beran, "Maximum likelihood estimation of the differencing parameter for invertible short and long memory autoregressive integrated moving average models," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 659–672, 1995.

[5] J. Yang, M. N. Nguyen, P. P. San, X. L. Li, and S. Krishnaswamy, "Deep convolutional neural networks on multichannel time series for human activity recognition," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[6] S. Song, V. Chandrasekhar, B. Mandal, L. Li, J.-H. Lim, G. Sateesh Babu, P. Phyo San, and N.-M. Cheung, "Multimodal multi-stream deep learning for egocentric activity recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 24–31, 2016.

[7] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in Neural Information Processing Systems*, pp. 802–810, 2015.

[8] T. Zufferey, A. Ulbig, S. Koch, and G. Hug, "Forecasting of smart meter time series based on neural networks," in *Workshop Data Analytics for Renewable Energy Integration (DARE), European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Riva del Guarda*, pp. 19–23, 2016.

[9] S. Zhang, S. Bahrampour, N. Ramakrishnan, and M. Shah, "Deep symbolic representation learning for heterogeneous time-series classification," *arXiv preprint arXiv:1612.01254*, 2016.

[10] V. Vapnik, "Pattern recognition using generalized portrait method," *Automation and remote control*, vol. 24, pp. 774–780, 1963.

[11] A. Aizerman, E. M. Braverman, and L. Rozoner, "Theoretical foundations of the potential function method in pattern recognition learning," *Automation and remote control*, vol. 25, pp. 821–837, 1964.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[13] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[14] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European Conference on Computer Vision*, pp. 740–755, Springer, 2014.