Machine Learning Techniques for Predictive Maintenance

Posted by Srinath Perera, Roshan Alwis, reviewed by Srini Penchikala on May 21, 2017.
Estimated reading time: 11 minutes **NOTICE:** QCon.ai - Applied AI conference for Developers Apr 9-11, 2018, San Francisco. Join us! 15 Discuss

A note to our readers: You asked so we have developed a set of features that allow you to reduce the noise: you can get email and web notifications for topics you are interested in. Learn more about our new features.

Key Takeaways

- Learn about Predictive Maintenance Systems (PMS) to monitor for future system failures and schedule maintenance in advance
- Explore how you can build a machine learning model to do predictive maintenance of systems
- Machine learning process steps like the model selection and the removal of Sensor Noises Using Auto-Encoders
- How to train the machine learning model and run the Model with WSO2 CEP product
- Sample application using NASA engine failure dataset to predict the Remaining Useful Time (RUL) with regression models

Uncover real-world practices of Machine Learning & Artificial Intelligence at QCon.ai Conference

April 9 - 11 2018, San Francisco.

Everyday, we depend on many systems and machines. We use a car to travel, a lift go up and down, and a plane to fly. Electricity comes through turbines and in a hospital machine keeps us alive. These systems can fail. Some failures are an just an inconvenience, while others could mean life or death.

When stakes are high, we perform regular maintenance on our systems. For example, cars are serviced once every few months and aircrafts are serviced daily. However, as we will discuss in detail later in this article, these approaches result in resource wastage.

Predictive maintenance predicts failure, and the actions could include corrective actions, the replacement of system, or even planned failure. This can lead to major cost savings, higher predictability, and the increased availability of the systems.

Predictive maintenance savings come in two forms:

- Avoid or minimize the downtimes. This will avoid an unhappy customer, save money, and sometimes save lives.
- Optimize the periodic maintenance operations.

To understand the dynamics, let's consider a taxi company. If a taxi breaks down, the company needs to pacify an unhappy customer, send a replacement, and both the taxi and driver will be out of service while in repair. The cost of failure is much higher than its apparent cost.

One way to deal with this problem is to be pessimistic and replace fallible components well before failures. For example, regular maintenance operations, such as changing engine oil or replacing tires, handle this. Although regular maintenance is better than failures, we will end up doing the maintenance before it's needed. Hence, it is not an optimal solution. For example, changing the oil of a vehicle for every 3000 miles might not use oil effectively. If we can predict failures better, the the taxi can go few hundred miles without replacing oil.

Predictive maintenance avoids both the extremes and maximizes the use of its resources. Predictive maintenance will detect the anomalies and failure patterns and provide early warnings. These warnings can enable efficient maintenance of those components.

In this article we will explore how we can build a machine learning model to do predictive maintenance. The next section discusses machine learning techniques, while the following discusses a NASA data set that we will use as an example. Sections four and five discuss how to train the machine learning model. The Section "Running the Model with WSO2 CEP" covers how to use the model with real world data streams.

Machine Learning Techniques for Predictive Maintenance

To do predictive maintenance, first we add sensors to the system that will monitor and collect data about its operations. Data for predictive maintenance is time series data. Data includes a timestamp, a set of sensor readings collected at the same time as timestamps, and device identifiers. The goal of predictive maintenance is to predict at the time "t", using the data up to that time, whether the equipment will fail in the near future.

Predictive maintenance can be formulated in one of the two ways:

- Classification approach - predicts whether there is a possibility of failure in next n-steps.
- Regression approach - predicts how much time is left before the next failure. We call this Remaining Useful Life (RUL).

The former approach only provides a boolean answer, but can provide greater accuracy with less data. The latter needs more data although it provides more information about when the failure will happen. We will explore both of these approaches using the NASA engine failure dataset.

Turbofan Engine Degradation Dataset

Turbofan engine is a modern gas turbine engine used by the NASA space exploration agency. NASA has created the following data set to predict the failures of Turbofan engines over time. The data set is available at PCoE Datasets.

The data set includes time series for each engine. All engines are of the same type, but each engine starts with different degrees of initial wear and variations in the manufacturing process, which is unknown to the user. There are three optional settings that can be used to change the performance of each machine. Each engine has 21 sensors collecting different measurements related to the engine state at runtime. Collected data is contaminated with sensor noise.

Over time, each engine develops a fault, which can be seen though sensor readings. The time series ends some time before the failure. Data includes unit(engine) number, time stamps, three settings, and readings for 21 sensors.

The following Figure 1 and 2 show the subset of the data.

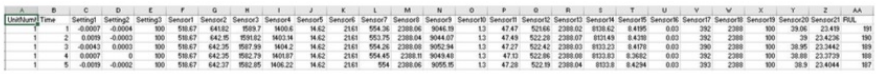

Figure 1: Subset of Data

A	B	C	D	E	F	G	H	I
UnitNumbe	Time	Setting1	Setting2	Setting3	Sensor1	Sensor2	Sensor3	Sensor4
1	1	-0.0007	-0.0004	100	518.67	641.82	1589.7	1400.6
1	2	0.0019	-0.0003	100	518.67	642.15	1591.82	1403.14
1	3	-0.0043	0.0003	100	518.67	642.35	1587.99	1404.2
1	4	0.0007	0	100	518.67	642.35	1582.79	1401.87

Figure 2: First few columns from Subset of Data

The objective of this experiment is to predict when the next failure will occur.

Predicting Remaining Useful Time (RUL) with Regression

While predicting RUL, the goal is to reduce the error between the actual RUL and the predicted RUL. We will use Root Mean Squared Error since it penalizes large errors severely, which will force the algorithm to forecast RUL as close as possible.

$$\text{Root Mean Squared Error (RMSE)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \quad y_i = \textit{predicted value},\ \hat{y}_i = \textit{actual value}$$

Phase 1: The following pipeline depicts the prediction process. As the first step to get a high level understanding about what's possible, we ran the pipeline with only highlighted steps. This runs the algorithm on raw data without any feature engineering.

Phase 1: Model Selection

The following figure 3 shows the Predictive Maintenance Pipeline for Model Selection. Here, only dark colored steps of the pipelines are used.

Figure 3: Predictive Maintenance Pipeline for Model Selection

We have used a wide range of regression algorithms available from scikit learn and H2O. For deep learning, we have used H2O Deep-Learning algorithm, which can be used in both classification and regression applications. It's based on multi layered feed forward neural network that is trained with stochastic gradient descent using back-propagation.

The following Figure 4 shows the results. Models could produce RMSE about 25-35, which means that the RUL will have an error of about 25-35 time steps.

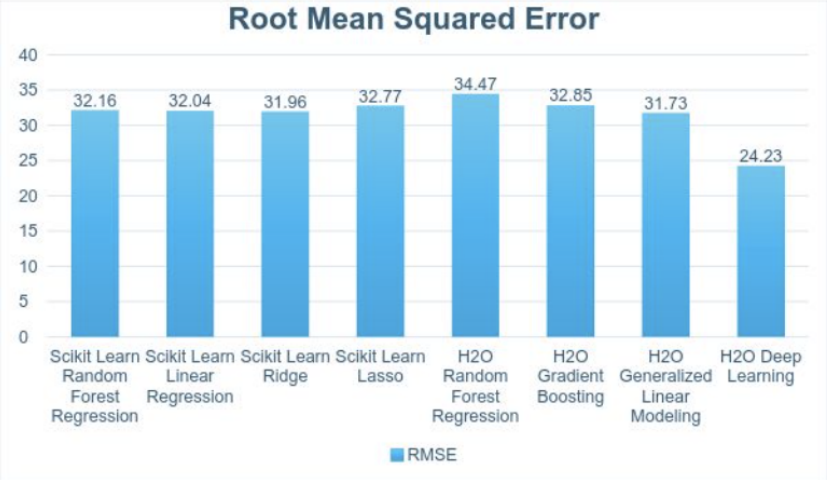

Figure 4: Root Mean Square Errors for Model Selection

For the next steps, we will focus on the deep learning model.

Phase 2: Removal of Sensor Noises Using Auto-Encoders

The following figure 5 shows the Predictive Maintenance Pipeline with Noise Removal. Here, only dark colored steps of the pipelines are used.

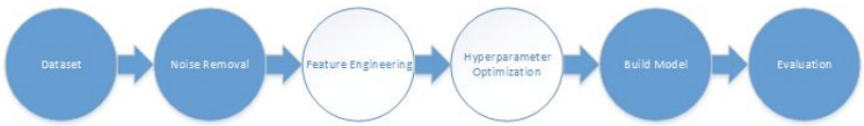

Figure 5: Predictive Maintenance Pipeline for Model Selection

Often, sensor readings have noise. The ReadMe file included with the distribution confirms this. Therefore, we removed noise using autoencoders. Autoencoder is a simple neural network trained with the same dataset as both the input and output of the network, where the network has fewer parameters than the dimensions in the data set. This works very similarly to the Principal Component Analysis (PCA) (http://setosa.io/ev/principal-component-analysis/) where data is represented in terms of its principal dimensions. Since noise has much higher dimensions than regular data; this process reduces the noise.

We used H2O Auto-encoder with three hidden layers and the following criteria to remove noise.

Removing noise improved the RMSE by 2.

	Without removing noise	After removing noise
RMSE	24.23	22.24

Table 1: RMSE with and without Noise Removal

Phase 3: Feature Engineering

The following figure 6 shows the Predictive Maintenance Pipeline with Feature engineering. Here, only dark colored steps of the pipelines are used.

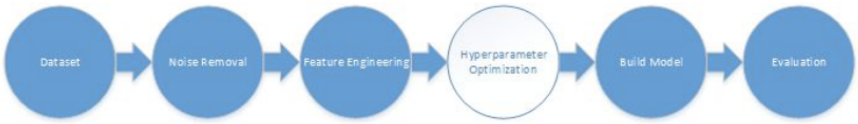

Figure 6: Predictive Maintenance Pipeline for Model Selection

In this step, we tried many features and kept the subset of features that are most predictive. Our dataset is a time-series dataset, hence readings are auto-correlated. Therefore, it's likely that prediction at time "t" is affected by some time window before "t". Most features we used are based on these time windows.

As discussed in section 3, the data set includes 21 sensors readings. We can find more details about readings from the ReadMe file provided with the data set. After tests, we only used data from sensors 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 15, 17, 20, and 21. For each sensor selected we generate features by applying moving standard deviation (Window of size 5), moving k-closest average (Window of size 5), and probability distribution with a window of size (Window of size 10).

Among other features that we tried but did not used in the final solution are: moving average, autocorrelation, histogram, moving entropy, and moving weighted average.

These features improved the RMSE by one.

	Without feature engineering	With feature engineering
RMSE	22.24	21.17

Table 2: RMSE with and without Feature Engineering

Phase 4: Optimizing Hyper-Parameters Using Grid Search

The following figure 7 shows the Predictive Maintenance Pipeline with Hyperparameters Optimizations. Here only dark colored steps of the pipelines are used.

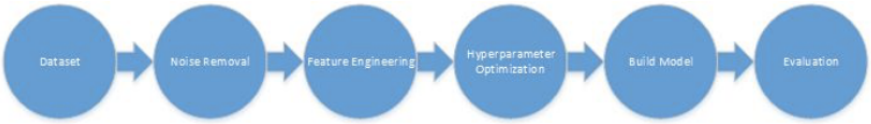

Figure 7: Predictive Maintenance Pipeline for Model Selection

Hyper parameters control the behaviour of the algorithm. At the final stage, we optimized the following hyper parameters epochs, distribution, activation, and hidden layer size. You can find a detailed description of each parameter from H20 Documentation. We used a grid search to find the best parameters, and the following table summarizes the results.

Model No	Hyper parameters: [epochs, distribution, activation, hidden]	Mean Square Error
69	[100.0, 'gamma', 'RectifierWithDropout', 200]	1738.03013686
13	[100.0, 'gamma', 'RectifierWithDropout', 100]	1768.44899187
70	[100.0, 'gamma', 'Maxout', 200]	1780.05468062
5	[100.0, 'gaussian', 'RectifierWithDropout', 100]	1790.78929369
67	[100.0, 'poisson', 'MaxoutWithDropout', 200]	1805.90205765
	Without hyper parameter optimization	With hyper parameter optimization
RMSE	21.17	18.77

Table 3: RMSE with different Hyperparameters

As the results depict, hyper parameter tuning improves the RMSE by about three. In the residual error histogram, you can see that the error is converging to "0". The frequencies of too early predictions and too late predictions are minimized.

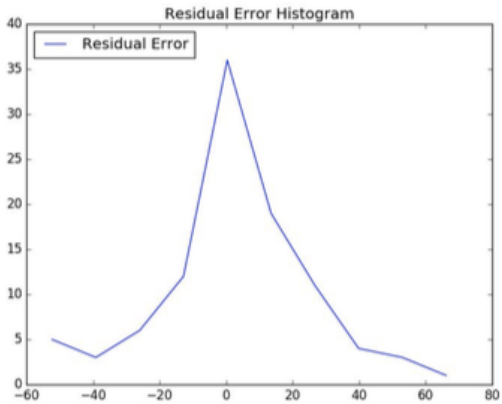

Figure 8: Predictive Maintenance Pipeline for Model Selection

Building a Model for Predicting Failure in Next N Steps

In this method, instead of providing the number of life cycles that remain, we are going to predict whether a machine will fail within the next 30 cycles. We will consider a failure as positive (P) and no failure as normal (N). We ran a deep learning classification model using the same feature engineering and noise removal process. Figure 9 depicts the results.

Confusion matrix

		Predicted	
		P	N
Actual	P	19	6
	N	0	75

Figure 9: Predictive Maintenance Pipeline for Model Selection

The accuracy describes what fraction of test cases are correctly predicted. It gives a ratio of the number of truly predicted test cases to all the test cases.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} = \frac{19+75}{19+75+0+6} = 94\%$$

Only considering accuracy can be misleading if the classes are not balanced. Class imbalance happens when the same classes are over-represented in the data set. With class imbalance, some models might have high accuracy but poor prediction performance.

To avoid this problem, we use precision and recall. Recall is the ratio between the number of positive values that have predicted and the number of positive values that should be predicted.

$$Recall = \frac{TP}{TP+FN} = \frac{19}{19+6} = 0.76$$

Precision is defined as the model's ability to predict positive values. It provides a ratio between the number of truly predicted positive values and the number of all predicted positive values that exist.

$$Precision = \frac{TP}{TP+FP} = \frac{19}{19+0} = 1$$

F1 score is a measure of the accuracy of the test. It takes both precision and recall to calculate the score.

$$F1\ Score = \frac{2 \times Precision \times Recall}{Precision+Recall} = \frac{2 \times 1 \times 0.76}{1+0.76} = 0.86$$

For accuracy, recall, precision and F1 score, it’s better to have their values close to 1.

Running the Model with WSO2 CEP

We built models in batch mode where we processed data stored in a disk. However, to apply a model, we need to feed data into a model at the runtime as the data becomes available. We refer to the processing of data as it comes in as "Stream processing". We used stream processing engine WSO2 CEP to apply the model.

We built the model using H20. H2o can export the model in one of the two formats: POJO (Plain Old Java Object) or MOJO (Model Object, Optimized). The former needs to be compiled and the latter can be used directly. We used MOJO model from CEP.

To evaluate the model, we used an extension in WSO2 CEP. WSO2 processes data that comes in a data stream using an SQL like query language.

As shown in the Figure 10, a Complex Event Processing system receives data as event streams and evaluates them against a set of SQL like queries. For example, a given query can calculate a one minute window over stockQuotes stream and join it with a one minute window over Tweets stream on a given condition, and send a event to PredictedStockQuotes stream.

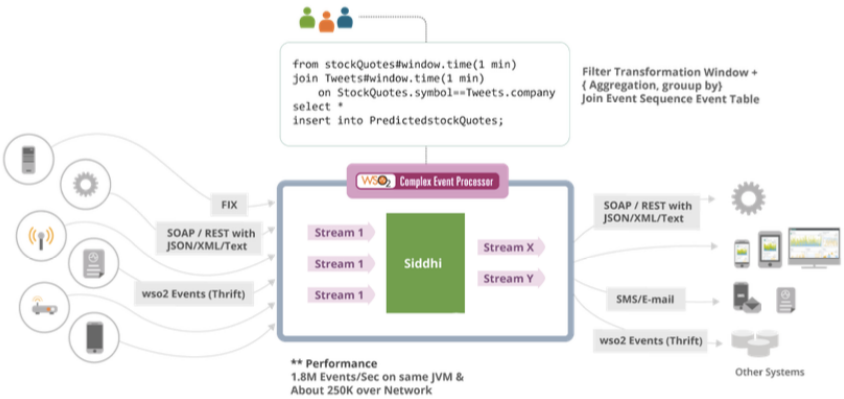

Figure 10: Predictive Maintenance Pipeline for Model Selection

A sample query to evaluate the model using CEP looks like following.

```
from data_input#h2opojo:predict('ccpp/DRF_model_python_1479702792496_1')

select T, V, AP, RH, prediction

insert into data_output
```

Following Figure 11 shows the overall pipeline that includes both training steps as well as evaluation steps.

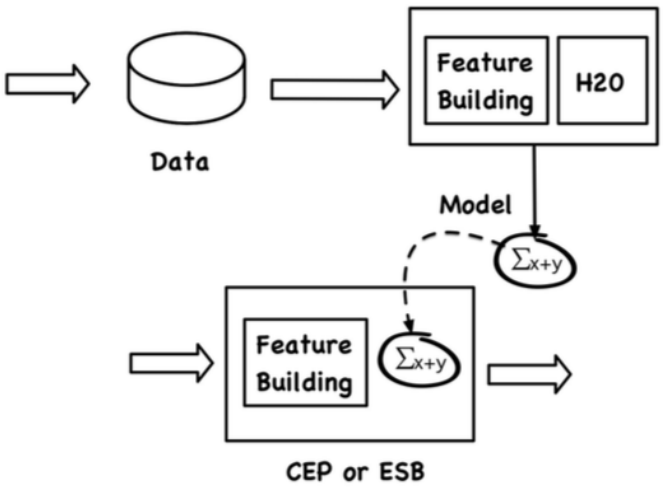

Figure 11: Predictive Maintenance Pipeline for Model Selection

The query takes events sent to stream "data input", and applies the machine learning model. Applying the machine learning model includes several steps:

1. Apply the preprocessing steps described in the "Phase 3: Feature Engineering" section to values in the event and create features
2. Evaluate the machine learning model using generated features
3. Return the results

Conclusion

The main objective of predictive maintenance is to predict when equipment failures can occur. Then prevent that failure by taking relevant actions. Predictive Maintenance System (PMS) monitors future failures and will schedule maintenance in advance.

This results in several costs savings:

- Reduces the frequency of maintenance.
- Minimizes the time spent on a particular equipment being maintained, and allows that time to be productive.
- Minimizes the cost of maintenance.

This article explored different approaches for predictive maintenance using different regression and classification algorithms. Furthermore, the article walked through many techniques for tuning those models. Our final solution predicted the RUL(remaining useful time) prediction with RMSE of 18.77 and predicted failure probability within the next N (30) steps with 94% accuracy.

The above techniques need significant amount of training data that include enough failure scenarios. Since failures are rare, data collections can take a long time. This remains a significant obstacle to applying predictive maintenance.

References

1. Deep Learning with H2O
2. PCoE Datasets
3. A Simulation Study of Turbofan Engine Deterioration Estimation Using Kalman Filtering Techniques
4. Predictive maintenance

About the Authors

Srinath Perera is a scientist, software architect, and a programmer that works on distributed systems. He is a co-founder of Apache Axis2 project, and a member of the Apache Software foundation. He is a co-architect behind WSO2's Complex Event Processing Engine. Srinath has authored two books about MapReduce and many technical articles. He received his Ph.D. from Indiana University, USA, in 2009.

Roshan Alwis is Batch Representative (CSE 13), Department of Computer Science and Engineering, University of Moratuwa.

Uncover real-world practices of Machine Learning & Artificial Intelligence at QCon.ai Conference