Final Report

Predicting Machine Failure Using Time Series Data

Contributors: Samuel Scovronski

Introduction

The data for this project was provided by Microsoft as part of their Azure Machine Learning sample experiments. It consists of hourly average telemetry data for 100 machines over 1 year, and their respective failure, maintenance, and error logs. This means there are 876,000 rows of telemetry data with a timestamp.

There are three groups that care about being able to predict failures or maintenance needs, but each group has the same common goals. The groups include manufacturing companies that use equipment in their production, companies that produce electronic or moving products, and consumers that use those electronic products. Their goals with these types of models are minimizing machine/product maintenance costs, minimizing unplanned downtime, and maximizing machine/product life. In other words, minimize the cost of ownership and the inconvenience of unplanned downtime

Problem Statement

Manufacturing companies want to be able to know when their manufacturing equipment will break down before it happens, while simultaneously minimizing the amount of time spent on maintenance of the equipment. When using the traditional preventative maintenance method for failure prevention, depending on the schedule, the company could be servicing the equipment too frequently or not frequently enough because it does not account for the natural variance between machines. If the machine is maintained too frequently the company will spend more time servicing the equipment, spend more on materials, and reduce the amount of time it could be used to produce more products. If not maintained enough, then some machines could experience a failure resulting in unplanned downtime and potentially causing a chain of failures in other parts resulting in more machine down time and more costly repairs.

Data Wrangling

The dataset provided by Microsoft contained 876,000 rows and 6 columns of telemetry data in a csv format. No information was provided about whether the data was generated or recorded from actual machines. Four of the columns consist of hourly average sensor data, while the other two columns mark the machine the data was recorded from, and a time stamp of when the data was recorded. There are also 4 additional csv files, that consist of maintenance logs, failure logs, error logs, and some basic metadata of each machine.

Upon initial inspection of the data, there were no major issues identified. There were no missing telemetry values or timestamps for every machine throughout the year. However, some of the column data types had to be converted and it is unbalanced. All timestamps had to be converted from a string to a DateTime, and the errorID values needed to have the non-numerical values stripped so the error number could be converted to an integer. With all issues resolved, the telemetry data needed to be transformed into rolling windows, since using a single instance of raw values is typically not strong enough to predict future failures.

Exploratory Data Analysis

During exploratory data analysis

Surprisingly, the computer age and model number seem to have no impact on the number of errors or failures that it experiences. In fact, there is no correlation between age and the number of failures, and on average a computer only experiences 0.12 more errors for every additional year old it is.

Figure 1: Number of Machine Failures vs. Machine Age

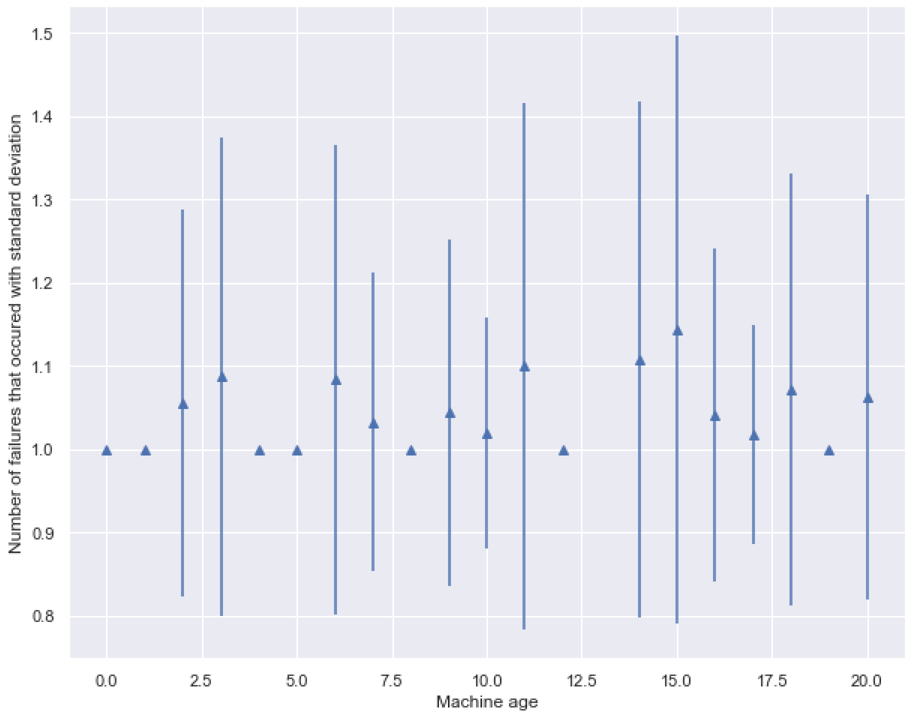
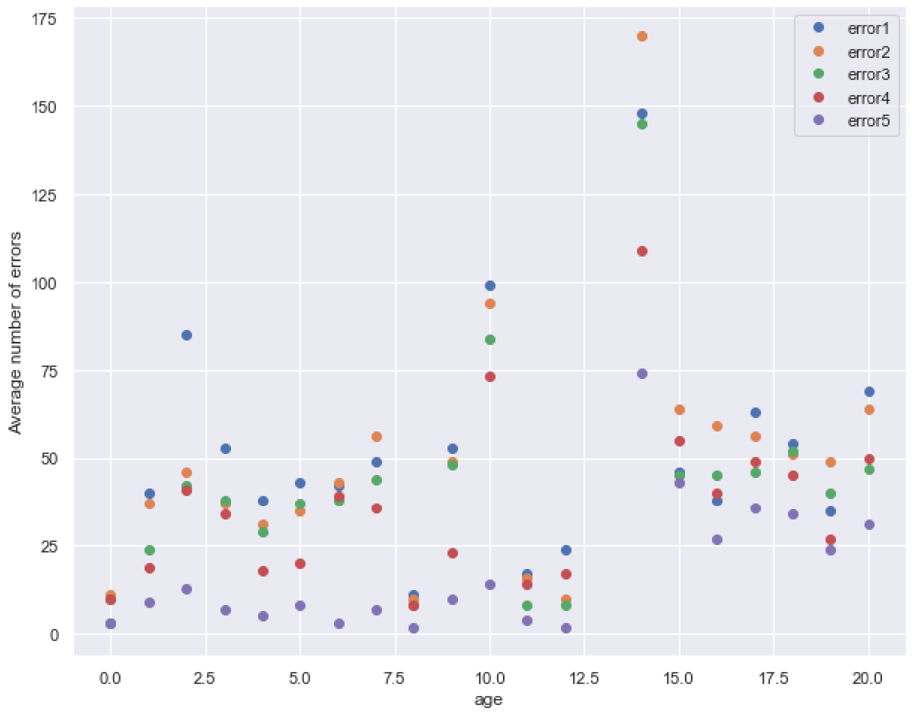


Figure 2: Count of each Error Type vs. Machine Age



Looking at the raw sensor data for individual machines there are noticeable spikes in the data, that become even more pronounced when the moving average is taken. Below are some graphs depicting the moving average of the telemetry data for a single machine, with varying window sizes. This begs the question, do these spikes occur before a failure, after a failure, or are failures not associated with these spikes. Overlaying the failure occurrences (orange dots) over the raw data we see that some of the failures occur near the time when the spikes occur, but it does not occur consistently.

Figure 3: Machine 99 Moving Average Telemetry Data

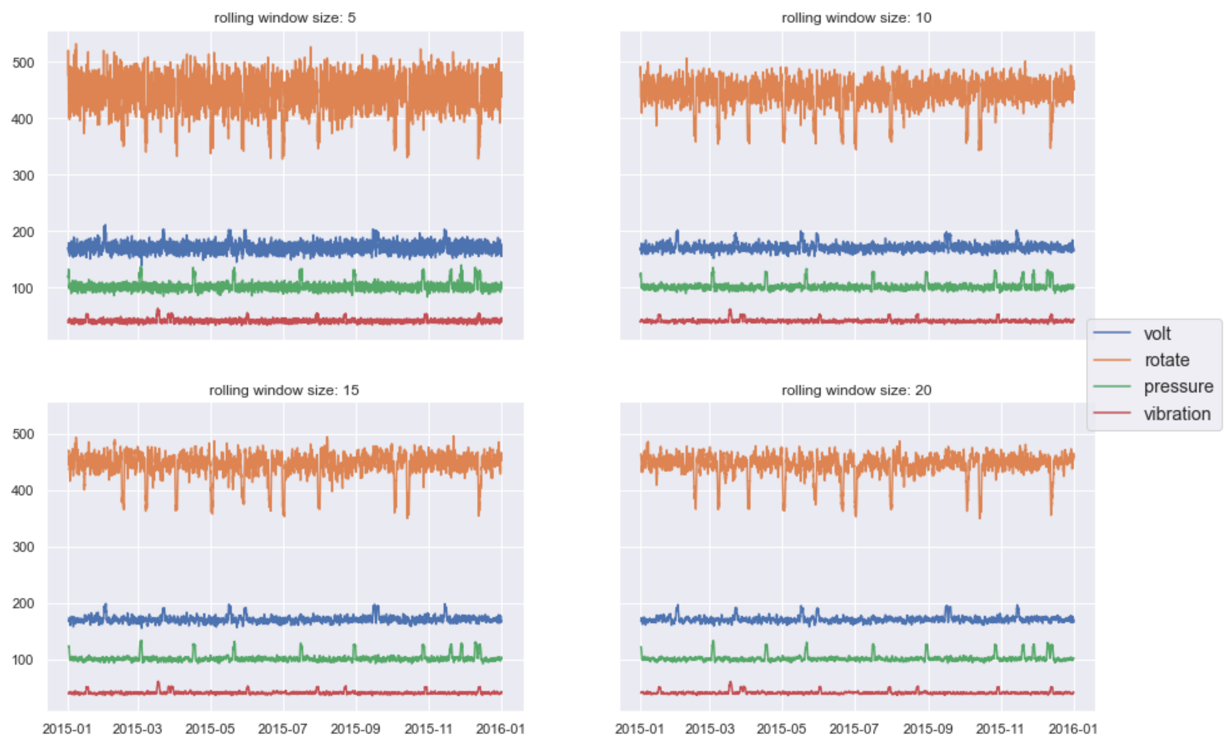
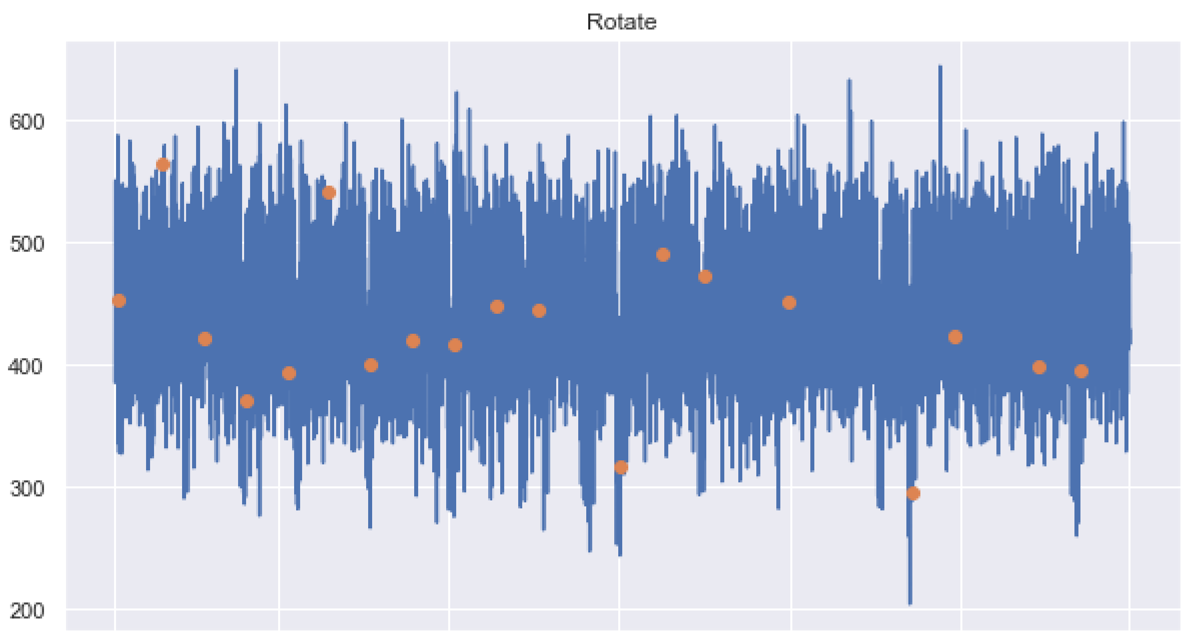
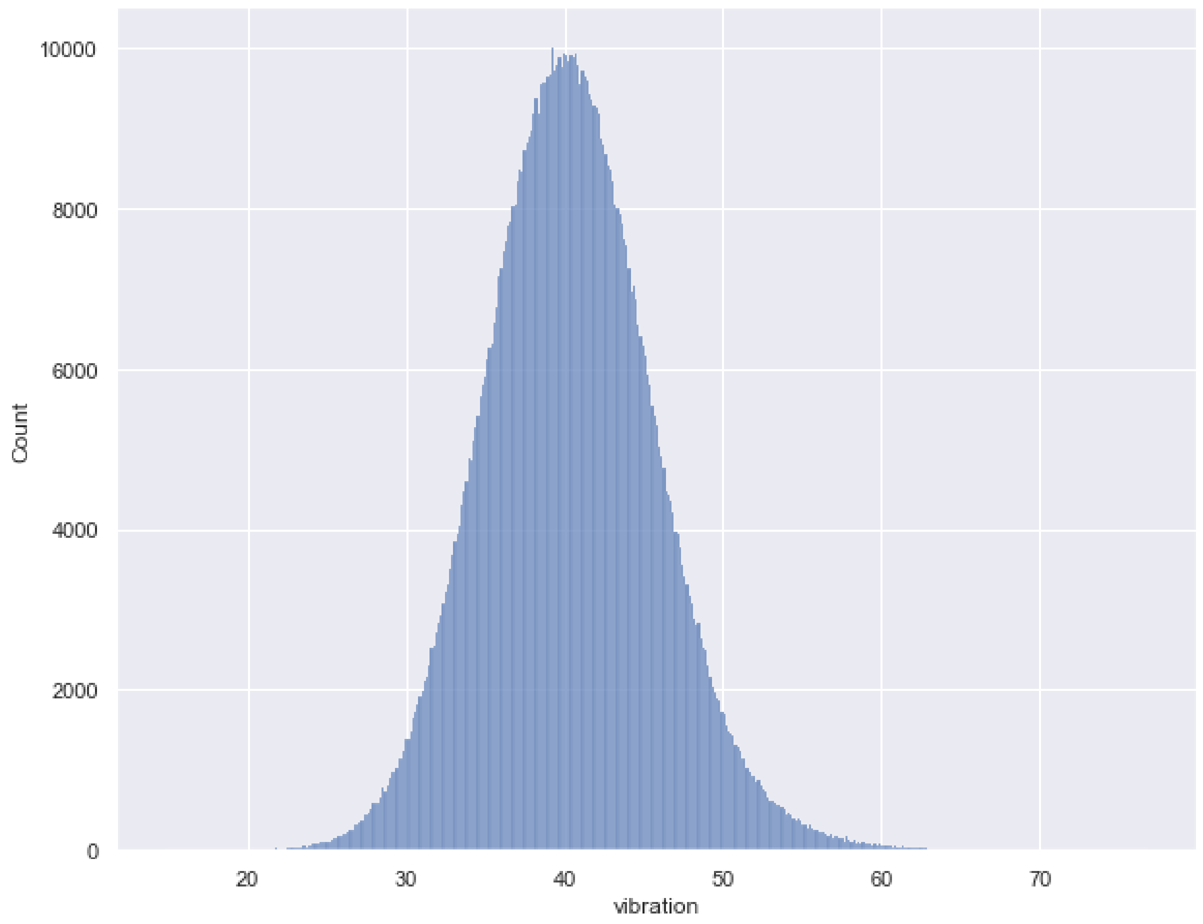


Figure 4: Raw Rotation Data for Machine 99 Overlaid with Failure Occurrences



Although it was not explicitly stated if the data was generated or measured on actual machines, it seems likely that the data was generated. The reason for the suspicion is that the data seems to be nearly perfectly normally distributed. The histogram below displays the distribution of all vibration values recorded for every machine.

Figure 5: Histogram of all Vibration Data



Modeling Setup & Results

Data Preprocessing

One of the first preprocessing steps typically required for time series data is ensuring that the timestep between samples is constant. This requires resampling of the data and then using an interpolation method, such as forward fill, linear interpolation, etc. If you do not want to have a constant timestep then you must divide your values by the timestep to get the instantaneous rate of change of your values.

The next typical preprocessing step for time series data is the windowing of the data, in which the past n rows of data are grouped together. The data is windowed because using instantaneous values is less helpful for predicting the future than using a couple of the most recent values. For this model, I used a constant step size between windows of 1, or in other words, the window was moved forward by one timestep for each window. Once the data is grouped into multiple windows, there is the option to either use statistical values or the raw values of each window as the features fed into the model.

Once the features are created, then the data needs to be normalized and split into training and testing data. For this project the data was standardized, and then it was split 70/30 into training and test data, respectively.

Model Criteria and Baseline Model Results

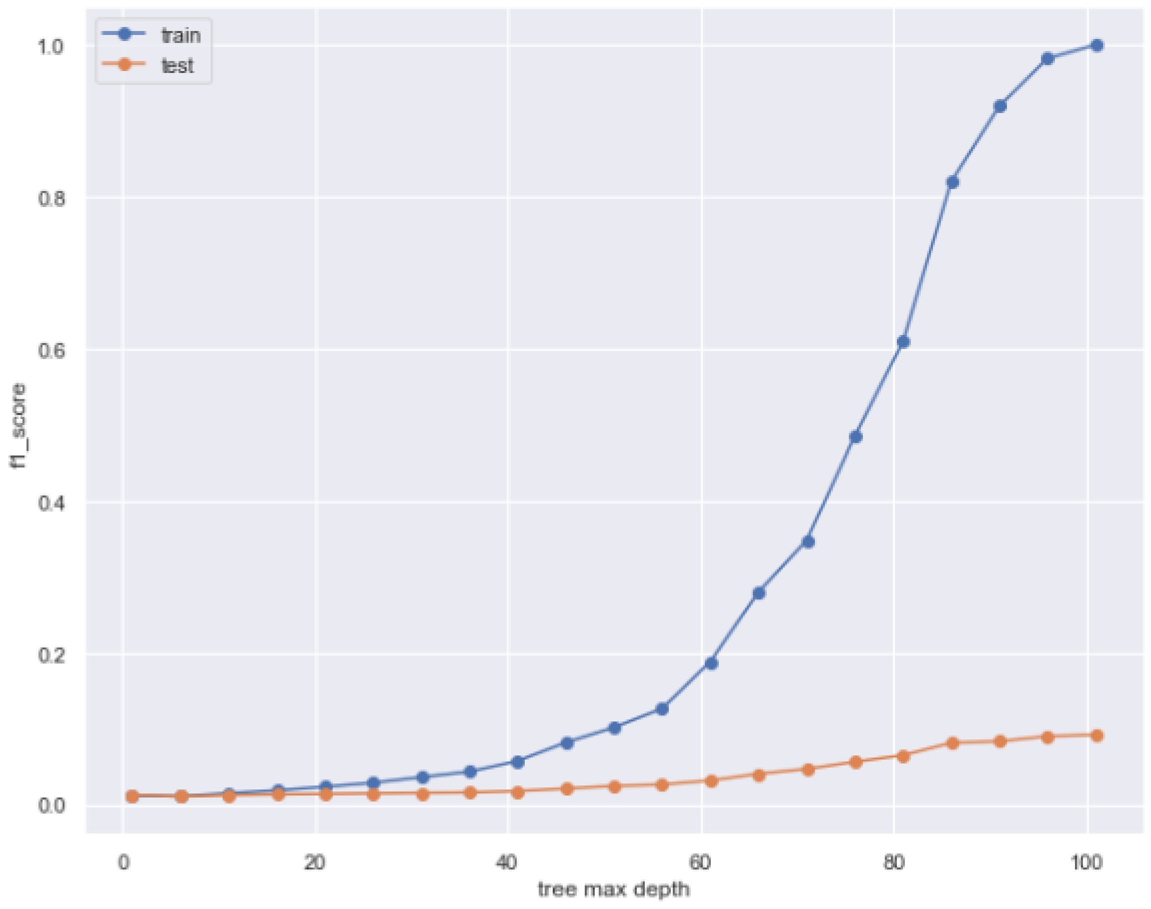
The overall goal for this model is to be able to always predict when a failure will occur without too much nuisance tripping. Having a bias toward getting 100% of the true positives, would mean that we want to have a high recall for failure. However, since nuisance tripping needs to be avoided a high precision is also required. Having high precision and recall can be summarized into a single number through the f1 score, which needs to be maximized. A high f1 score means that when the model flags a potential failure, the user can feel confident that there will be a failure. For this project, the goal is to achieve an f1 score greater than 0.9.

Due to the nature of the problem and how few failures there are (700 in total), there are not many good choices for a baseline model. Obviously, predicting a failure will occur every hour would result in 100% recall, but a precision of 0. This would not be helpful, since the user would learn to ignore any warnings. However, to get an idea of a worst-case scenario a couple of random guess models were created to determine the distribution of f1 scores. Using random guessing, the average f1 score was approximately 0.00175.

Model Selection

During the model selection process, simpler models were evaluated first, and then the complexity was increased. With this in mind, a decision tree model was evaluated first by making multiple models of varying tree depth to identify a model that was not underfitted or overfitted. This process was done with both the raw telemetry values and statistical features to determine if there was a difference. The statistical values that were calculated for each window include: minimum, maximum, mean, sum, standard deviation, skew, and kurtosis. While modeling the error was weighted according to the number of each timestamp that experienced a failure or not due to our data being unbalanced. Moving to this slightly more advanced modeling method resulted in a 55x improvement in the f1 score over the baseline model, but the overall score was still not very good, coming in at approximately 0.091. The model complexity was then stepped up by looking at Random Forest, Logistic Regression, and Support Vector algorithms to increase the f1 score. However, this additional complexity resulted in no improvement and left the only option of using a neural network classifier, or taking an entirely different approach.

Figure 6: Decision Tree Classifier f1 Score vs. Tree Depth



Neural Net Classification

Before the neural net can be created, some additional data preprocessing must be completed. Because the neural net algorithms are not set up to weigh the errors anyway but equal, the data needs to be oversampled. Oversampling comes with its own set of problems especially when the imbalance is as significant as in this case. The first issue to address is which method to use for oversampling. There is the option to resample by making exact duplicates of the rows, but then there is the risk of overfitting to those exact values. Alternatively, the SMOTE (Synthetic Minority Over-sampling Technique) method could be used, which generates new data based on the values of the under-represented class. The downside to this method is that the values created might not be realistic and depending on how similar the values are to the over-represented class it could make the two classes more similar. The other issue is making sure there is no contamination between your training and testing data sets. If over-sampling is performed before splitting of the data then there could be the same values in both sets (resampling), or created values being too similar (SMOTE). For this project the SMOTE method was used.

The first Neural Network model that was used is scikit learns MLP classifier. The MLP classifier algorithm does training and testing for overfitting by breaking the data provided into two groups. However, since all of the values were generated the neural net quickly overfitted the training data resulting in very different scores between the training and validation data (below). On the bright side, the model is now able to achieve an f1 score greater than 0.1, which is much closer to the goal.

Dataset	Precision (Fail)	Recall (Fail)	F1 Score (Fail)	Accuracy
Train	0.99	1.00	1.00	1.00
Validation	0.97	0.21	0.35	0.60

To avoid overfitting, the test data needs to be generated and specified separately from the training data. This can be done by creating a Neural Network with the keras package. Before the model was created the data was broken into three sets, training, testing, and validation. Then the SMOTE over-sampling method was performed on all sets individually to prevent contamination from the other datasets. The model then was fit against the training data and tested against our training test set over 100 epochs, during which the model that performed best on the test set was saved. Using this method there is less discrepancy in performance between the test scores and the validation scores, simultaneously resulting in a f1 score near 0.9.

Dataset	Precision (Fail)	Recall (Fail)	F1 Score (Fail)	Accuracy
Test	0.94	0.83	0.88	0.89
Validation	0.94	0.86	0.90	0.90

Future Improvements

Although it was possible to predict failure 5 hours in the future with relatively high accuracy, it would be beneficial to extend the prediction time further. Ideally, this would be manifested by creating a trade-off graph showing how the f1 score is impacted by the time to failure. The multiple models created could then be utilized to create a remaining useful life prediction, giving a probability of failure within X amount of time.