Larry Rabang, Jack Saele, Saul Varshavsky, Brian Nielsen, Gabriel D'Alessio

Table of Contents

I. Introduction

A. Business Problem

The purpose of this project was to investigate which turbine sensor readings and environmental conditions are more likely to result in the occurrence of a fault code within 24 hours of the time that a given turbine sensor reading or environmental condition was recorded.

B. Importance of Business Problem

Understanding this problem is important because the answers may be a strong indication of multiple things. First, it may indicate that some areas of a turbine need extra care when being designed, or that prioritizing the design of certain areas over others can produce better results. Additionally, it could indicate that certain environmental conditions tend to negatively affect turbines. This may be a sign that the engineering techniques behind the turbines should be re-evaluated, so that the turbines can better withstand these certain environmental conditions.

By using predictive modeling techniques, it could be possible to detect fault codes before they occur. This would be important to Berkshire Hathaway Energy (BHE), as it would allow the company to run new data through a predictive model, which would then advise BHE on whether or not that new data results in certain turbines experiencing a fault code within the next 24 hours. By using a predictive model to look forward 24 hours into the future, BHE could take preventative measures (like temporarily turning off a turbine that is predicted to fault), allowing the company to prevent unnecessary costs associated with turbine failures.

C. Data Used

Note that all code files used in this white paper can be found in the folder titled Finalized Code.

The data used to address this problem is the sensor readings and environmental conditions (weather data). The specific sensor readings used are Fault Codes, Generator RPM,

Gearbox IMS Bearing Temperature, Gearbox HS Bearing Temperature, Ambient Temperature, Gearbox Oil Temperature, Active Power, and Hydraulic Pressure. The weather data columns used for addressing this problem are date, max_tmpf (maximum temperature in degrees Fahrenheit), min_tmpf (minimum temperature in degrees Fahrenheit), precip (rainfall in inches), max_gust (maximum wind speed gust in miles per hour), min_rh (minimum relative humidity as a percent), max_rh (maximum relative humidity as a percent), max_dwpf (max dew point temperature in degrees Fahrenheit), min_dwpf (min dew point temperature in degrees Fahrenheit), and avg_sknt (average wind speed in knots). Since we knew that the turbines are located somewhere in Iowa (but not where in Iowa), we used weather readings from Des Moines to represent Iowa environmental conditions as a whole. In addition, this weather data was recorded on a daily basis, so all sensor readings on the same day will be associated with the same weather data recorded for that day, regardless of turbine or time of day a sensor reading occurred.

II. Data and Exploratory Analysis

A. Data Cleaning and Preparation

Note that the code file for cleaning and merging the sensor readings data is titled "initial_loading_and_cleaning.R". Additionally, the code file for cleaning and merging the weather data is titled "merging_weather.R". Finally, the code file for imputing missing values is titled "imputing.R". Those code files are in the folder titled "Finalized Code".

In order to use the data described in the subsection titled "Data Used" (discussed in the section titled "Introduction"), we needed to clean it first. After taking a closer look at the sensor readings data sets, we realized that all data sets were of similar structure. We cleaned each of the sensor readings data sets using similar for loops, making our data cleaning code reproducible; we

could easily clean any new data provided later. For the weather data, we noticed it was mostly initially clean; we simply removed columns unnecessary for addressing the business problem.

After the initial cleaning, we aggregated all the data into 30 minute intervals in order to merge the data. We called the new time variable RoundedDateTime. Within each 30 minute time interval, we kept the minimum, maximum, average, and standard deviations of each sensor reading. This helped us understand the sensor readings in those 30 minute segments well, while also reducing the number of rows in the data frame. We then merged all the sensor readings data into one dataset using a composite key of turbine ID and RoundedDateTime, since there was no primary key. The weather data was merged to this data set using the date.

Next, we imputed all missing data using Kalman filtering. There were instances where, for example, there was Active Power data in a given 30 minute time interval, but there was no Hydraulic Pressure data in that same time interval. The Hydraulic Pressure data was missing. We considered multiple ways of imputing the missing values, such as using the mean or median, but we chose the Kalman filtering technique because it takes into account data trends. For example, if Hydraulic Pressure values have been slowly increasing over a time frame of 10 hours and there is a missing value at a certain date-time within that time frame, then Kalman filtering infers a realistic value to replace that missing value, so the trend of Hydraulic Pressure values increases.

B. Creation of Fault Code Events

Note that the code file used for creating fault code events is titled "fault_event_creation.R". Additionally, the code file used for merging the fault code events to the rest of the data is titled "combining_fault_events.R".

In the fault codes data, many fault code descriptions are repeated over and over again in short time intervals. In order to minimize these repeated values, we created fault code events, where each fault code event represents the start of a new set of occurrences of a fault code description. We used a 6 hour time frame to see if at least two instances of a certain fault code

description occurred within 6 hours of each other. If they did, we considered these instances to be part of the same fault code event; otherwise we considered them to be different fault code events.

To create fault code events, we first sorted the fault codes data by the turbine ID, then by fault code description, and finally by the date-time of a given fault code occurrence. Then, we looped through every row and check to see if the next row matched 3 conditions: does the turbine ID of the current row match the turbine ID of the previous row, does the fault code description of the current row match the fault code description of the previous row, and does the fault code description of the current row occur within 6 hours of the fault code description of the previous row. If all three conditions are met, then the end time of the current fault code event gets changed to the start time of the fault code event of the next row. But if any of the conditions are not met, we end the current fault code event (with an end time equal to the date-time of the current row); we then start a new fault code event with a start time equal to the date-time of the next row of fault codes data. We ended up with 1,733 fault code events, each one having a turbine ID, start time, fault code number, fault code description, fault code type, and end time. Based on information that we were provided, we knew that some of the fault code descriptions (while they influenced turbine stoppages) were unimportant. Because of this, we decided to remove fault codes that matched the following fault code descriptions: “Stopped Due To Power Up Delay”, “Local, Ad-Hoc / Repair Work”, “Stopped, Untwisting Cables”, and “Local, Scheduled Service Work”. As a result, this left us with 1,369 fault code events.

Next, we attached each row from the sensor readings data to the first fault code event that occurs for the same turbine after the date-time of a given sensor reading. We then calculated the difference in time (hours) between the date-time of the sensor reading and the start time of the fault code event. We stored that information in a variable called TimeUntilFault.

Finally, we created a binary variable called FaultCodeOccurs. It is 1 if TimeUntilFault is less than or equal to 24 hours; 0 otherwise. FaultCodeOccurs is our target variable for predictive modeling: it indicates whether a fault code occurs within 24 hours of a given sensor reading.

C. Exploratory Analysis

After cleaning the initial data provided by BHE, we noticed that a handful of the 11 turbines had incomplete sensor readings. This can be better understood when looking at the visual in Figure 1. The visual portrayed in Figure 1 drew much confusion to other data scientists. As a result, this prompted BHE to provide us with new data with more complete sensor readings data. However, the new data only had information on sensor readings from June 2, 2022 to September 2, 2022, so the predictive models we would later create have some limitations.

Please note that the code file used for exploratory data analysis is titled "Exploratory Data Visualizations.R" which can be found inside the folder Finalized Code.

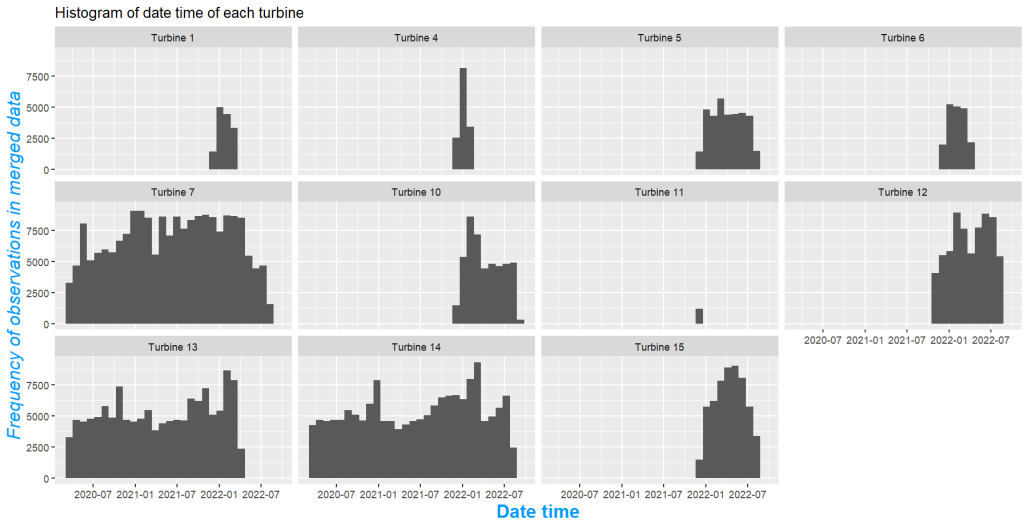

Figure 1: These histograms show the date-time recorded for all of the sensor readings corresponding to each of the 11 turbines in the original data, with the date-time being on the x-axis and the frequency of observations on the y-axis. As shown in the histogram, only turbines 7, 13, and 14 have sensor readings data for most of the date-time span.

III. Methods

A. Analytics Problem

Refer to the data file "final_dataset.csv" to see the merged data set when FaultCodeOccurs is created. This is our fully prepared dataset that we ended up using for predictive modeling.

Having our data set fully prepared to address our business problem, let's translate our business problem to reflect this: Which sensor reading columns and weather data columns will result in a fault within the next 24 hours (and therefore have a value of 1 for FaultCodeOccurs)?

B. Descriptive Modeling

Originally, we were going to utilize descriptive modeling in addition to predictive modeling to answer our business problem. However, brief analyses of various multivariate visualizations revealed that this would be inappropriate, as our nonlinear data would not fit well with any type of linear modeling. This is shown in Figure 2, 3 and 4. In nonlinear data, the relationship between the variables may be curved, and the effect of a one-unit change in a predictor variable may vary depending on the values of other predictor variables. Linear models may not capture these nonlinear relationships, leading to bad model performance and predictions.

Note that Figure 2,3 and 4 utilize code from the file "Final Data Visualizations.R".

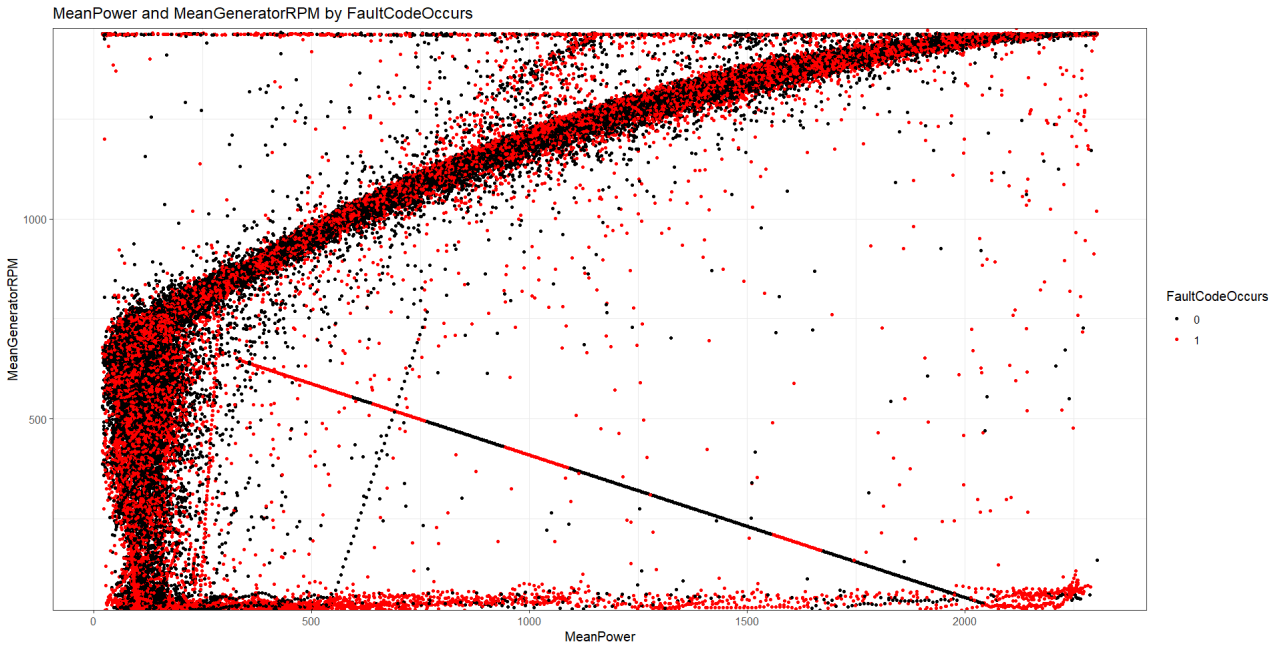

Figure 2: This figure shows the MeanPower on the x-axis and MeanGeneratorRPM on the y-axis in relation to FaultCodeOccurs being 0 or 1. There is a clear, nonlinear relationship.

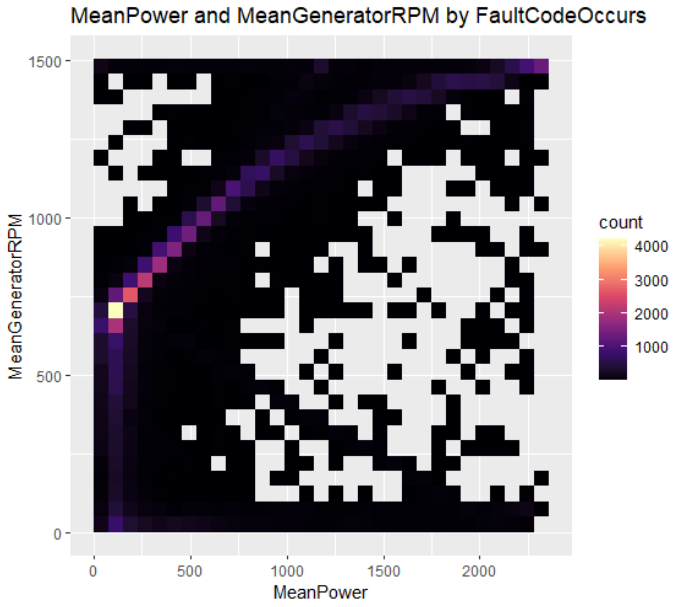

Figure 3: This figure shows the same variables as figure 2, but as a density plot. This more clearly shows a y=log(x) or y=sqrt(x) relationship, but it isn't a strong relationship.

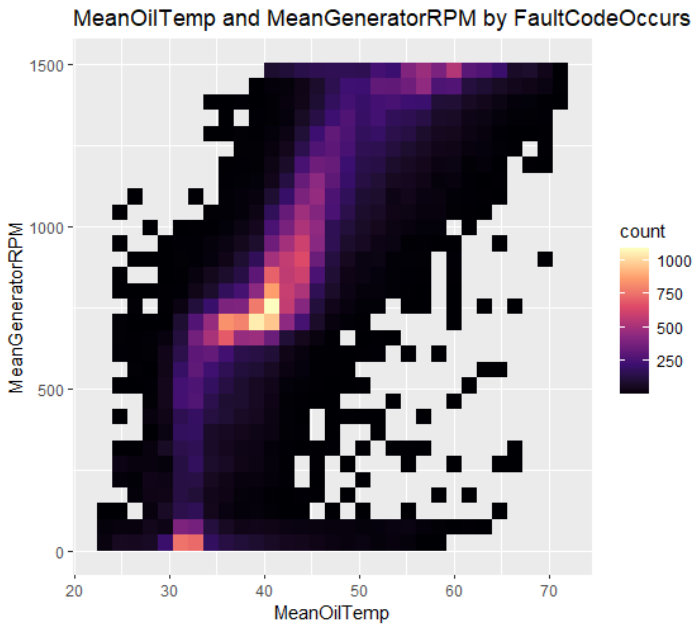

Figure 4: This figure shows the nonlinear relationship between MeanGeneratorRPM and MeanOilTemp. The exact relationship shown is relatively unclear.

C. Predictive Modeling Technique Chosen

We chose to create a random forest model to try and predict FaultCodeOccurs for two main reasons. Non-linear relationships between variables rule out the use of generalized linear models, and Random Forest Classifiers are great for binary target variables, which we have. We also considered creating a neural network or k-nearest neighbor algorithm to try and model whether there would be a fault in the next 24 hours. We chose not to pursue these methods, due to time constraints. We liked that random forests have some increased interpretability over neural networks, and that they can handle large amounts of data. Also, when creating a KNN model with large amounts of data, it requires a lot of processing power to run, which we did not have.

D. Random Forest Modeling with the Turbine ID

Note that the code file used for predictive modeling with the turbine ID is titled "PredictiveModelingWithTurbineId.R".

In creating our random forest models, we used the sensor readings and weather data as our predictor variables and FaultCodeOccurs as our target variable. In this case, we didn't focus on predictive modeling for sensor readings that applies to turbines as a whole, but rather to specific turbines. Hence, we used the turbine ID as a predictor variable. It's also important to note that we never used any of the columns from the fault codes data as our predictor variables, as that knowledge would not be known at the time of the sensor readings.

After specifying the predictor variables used for random forest modeling, we created a training dataset with 70% of the data, saving the other 30% for testing. Using the training data, we created a baseline random forest model with an m-try of sqrt(38), since we used 38 predictor variables. M-try is a parameter that determines how many variables are provided to each tree in the forest. An m-try that's too low means that we are not providing the model with enough information to make a good prediction, but an m-try that's too high results in the decision trees that comprise the random forest being too similar, resulting in overfitting. The OOB error rate for the baseline random forest was 5.56%. This means that 5.56% of the time when trying to predict a fault code occurrence, the baseline random forest makes a wrong prediction (predicting a 1 for FaultCodeOccurs when it should have been 0, or vice versa).

Next, we tuned the random forest, in order to see if the OOB error rate (refer to Figure 5) would decrease. A decreased OOB error rate means the random forest model is less likely to make a wrong prediction, increasing accuracy. We tuned the baseline random forest by creating a random forest for every m-try from 1 to 38. We then plotted the OOB error rate for each of the 38 forests (Figure 4) to see which had the lowest OOB error rate, and chose that forest.

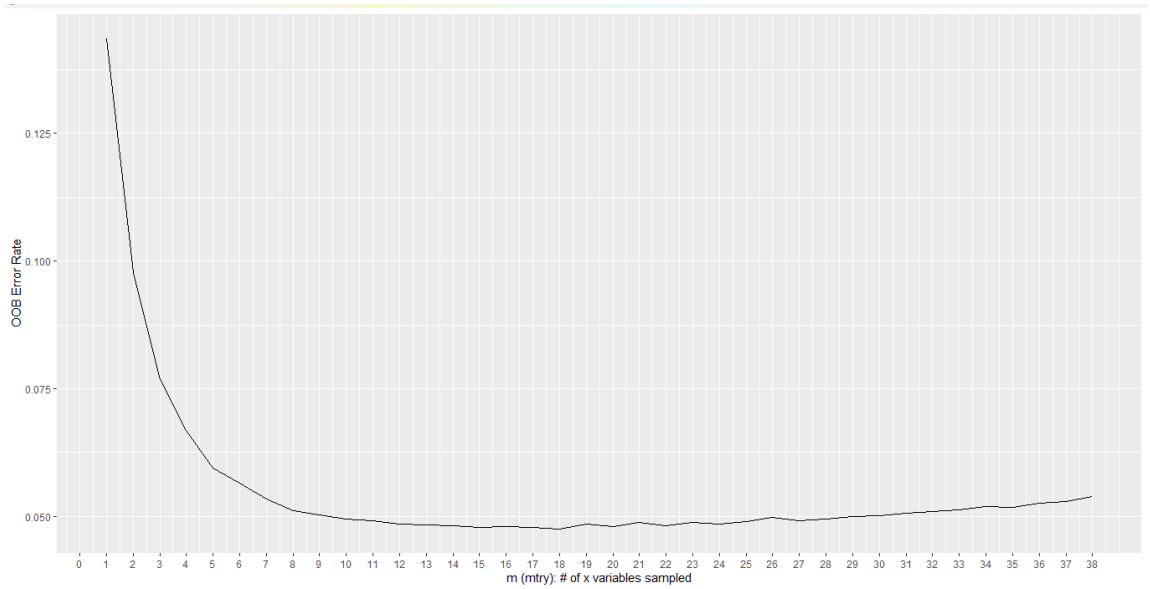

Figure 5: This is an OOB error rate graph. As shown, an m-try of 18 appears to have the lowest OOB error rate (for modeling with turbine ID).

Based on the results of the graph portrayed in Figure 5, we decided to use an m-try of 18, as it results in the lowest OOB error rate. Hence we created a new, finalized random forest model that uses an m-try value of 18. The OOB error rate for the new, finalized random forest is now 4.73%, which is smaller than the initial 5.56% OOB error rate. This means the new model now only makes a wrong prediction around 4.73% of the time.

E. Random Forest Modeling without the Turbine ID

Note the code file used for predictive modeling without the turbine ID is titled "ModelingWithoutTurbineId.R".

There is value in having a random forest model that is trained with the turbine ID. It will likely be more accurate than a random forest model made without turbine ID, as it creates a different model for each turbine, allowing us to know which turbines are more prone to failure in specific situations. However, a model trained with turbine ID as a predictor variable will only

know how to handle the specific turbines it is trained with, and BHE has more than just the 17 we were given. Because of this, we also trained a model without turbine ID as a predictor variable; this model would be more applicable to all of BHE's turbines. Other than turbine ID, this model was trained and tested on the same data as the first. After trying all possible values of m-try (now 37 instead of 38), the ideal value of m-try for this new model is 20, shown by Figure 6. This model has an OOB error rate of 9.98%, which is notably worse than the first model.

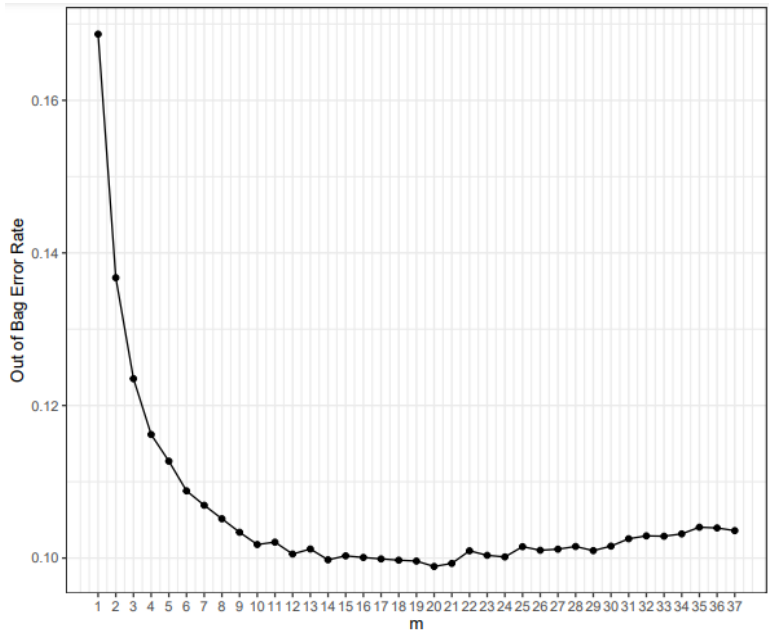

Figure 6: This is an OOB error rate graph for modeling without the turbine ID. As shown, an m-try of 20 appears to have the lowest OOB error rate.

IV. Results and Testing

A. ROC Curve for Random Forest Model with the Turbine ID

Note that the code file used for creating an ROC Curve with the turbine ID is titled "PredictiveModelingWithTurbineId.R".

We used the new, finalized random forest model with the turbine ID to create an ROC Curve that displays the optimal pi* to use for our testing dataset (which includes the predictions the model made). Random forest models do not output a 0 or a 1, but rather a probability of it being a 0 or 1. As a user of the model, you can choose what probability to use as your cutoff, or pi*, for what probabilities mean that you are predicting a fault or not. The higher pi* is, we are less likely to predict a 1 for FaultCodeOccurs. This results in less false positives, but more false negatives. An ROC curve shows the models performance at every possible value of pi*, and outputs which value is optimal. As shown in Figure 7, the optimal pi* value is 0.491. This means that if the probability of FaultCodeOccurs being a 1 is greater than 0.491 (pi*), then FaultCodeOccurs will predict a 1. This level of pi* has a specificity of 0.958 and sensitivity of 0.941. A specificity of 0.958 means that if a fault code does not occur within 24 hours of the date-time of a sensor reading, the model accurately predicts that 95.8% of the time. Meanwhile, a sensitivity of 0.941 means that if a fault code does occur within 24 hours of the date-time of a sensor reading, the model accurately predicts that 94.1% of the time.

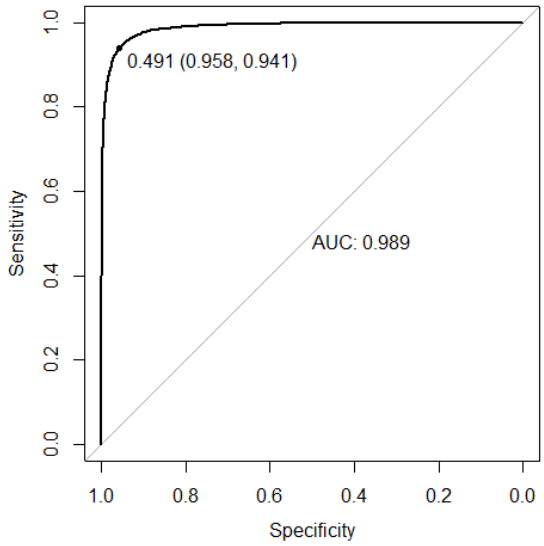

Figure 7: This ROC Curve corresponds to the finalized random forest (with turbine ID). The optimal pi* value is 0.491, corresponding to a specificity of 0.958 and a sensitivity of 0.941.

B. ROC Curve for Random Forest Model without the Turbine ID

Note that the code file used for creating an ROC Curve without the turbine ID is titled "ModelingWithoutTurbineId.R".

When using an m-try of 20 for the finalized random forest model without the turbine ID (refer to the subsection "Random Forest Modeling without the Turbine ID", in the section "Methods"), we get the ROC Curve shown in Figure 8. The ROC Curve shows that the optimal pi* is .418. At this level, when a fault code occurs, the model correctly predicts it 90.3% of the time; when a fault code doesn't occur, the model correctly predicts it 89% of the time.

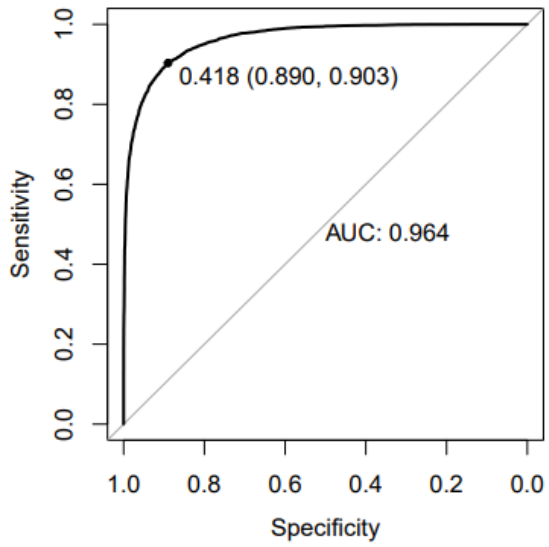

Figure 8: This ROC Curve (for the random forest model without turbine ID) shows an optimal pi* value of 0.418, corresponding to a specificity of 0.89 and a sensitivity of 0.903.

C. Variable Importance Plot for Random Forest Model with Turbine ID

Note that the code file used for creating a Variable Importance Plot with the turbine ID is titled "PredictiveModelingWithTurbineId.R".

After performing the steps outlined in the subsection titled "Random Forest Modeling with the Turbine ID" (which is inside the section titled "Methods"), we created a Variable Importance Plot, displaying the variables that have the biggest effect on predicting a fault event. The greater the Mean Decrease in Accuracy of a variable, the more impact that variable has on our predictions. As shown in Figure 9 (the Variable Importance Plot of the random forest model with turbine ID), turbine ID is by far the most important variable. In effect, the model is creating a bunch of smaller models for each of the turbines, which results in the overall model being more accurate than a model that does not have turbine information. While the model is more accurate for our data, it is difficult to know why. It could be because there are real differences between the turbines and what causes them problems. Or, it could be that the differences in the turbines are mostly noise, and the model is overfitted to be specific to this time period for those turbines. After the turbine ID, the variables average wind speed in knots, average ambient temperature, and standard deviation in ambient temperature have the biggest impact on the model. We don't know directly how they impact fault events (Does an increase in average ambient temperature increase the likelihood of a fault code occurrence?), which is one of our model's limitations.

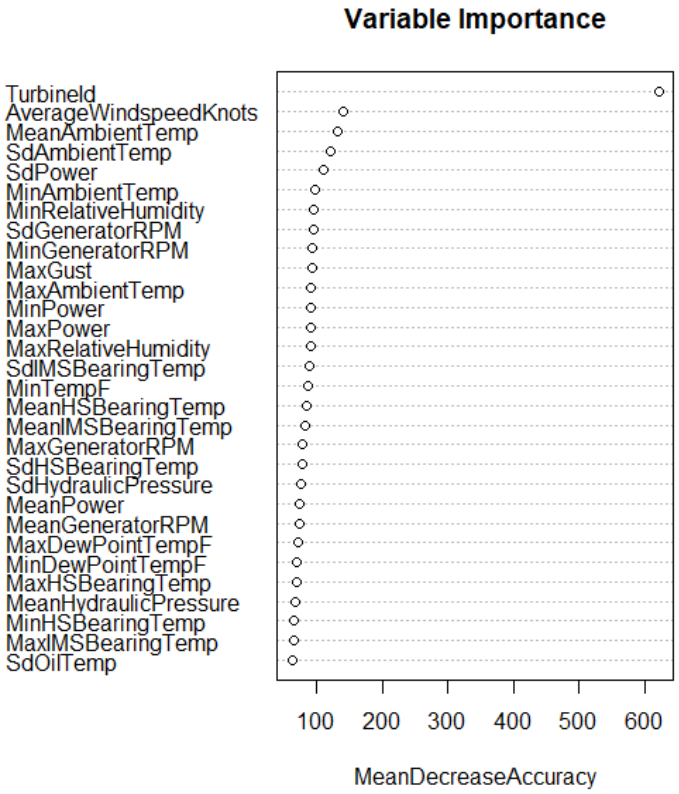

Figure 9: This is the Variable Importance Plot for the model with TurbineId.

D. Variable Importance Plot for Random Forest Model without Turbine ID

Note that the code file used for creating a Variable Importance Plot without the turbine ID is titled "ModelingWithoutTurbineId.R".

We also made a Variable Importance Plot for the model without the turbines. The results are similar to those with the turbines (as shown in Figures 9 and 10), with MeanAmbientTemp, AverageWindspeedKnots, and SdAmbientTemp all being among the most important variables, but the model without the turbines has SdHydraulicPressure as the most important variable. We do not know why it goes from somewhat important to the most important, but it is peculiar.

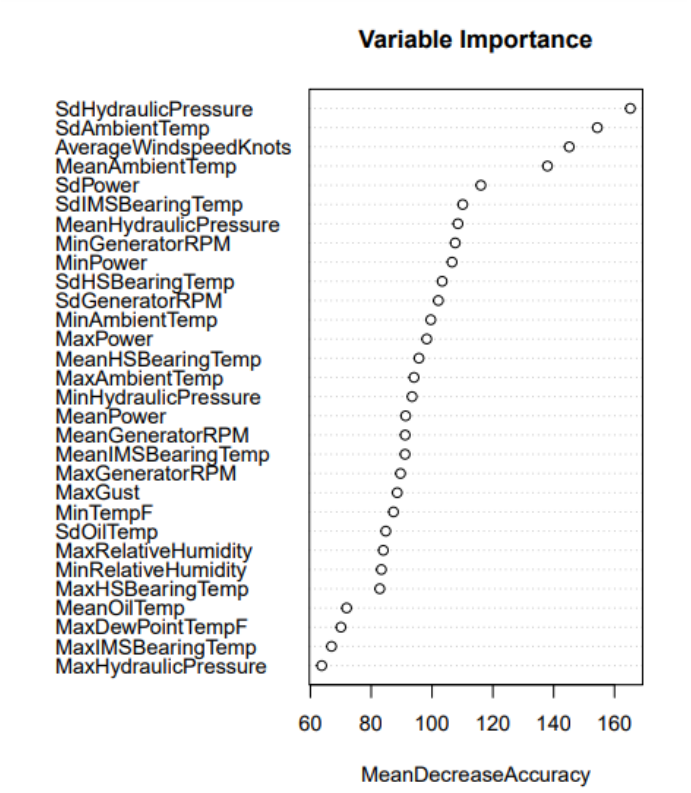

Figure 10: This is the Variable Importance Plot for the model without TurbineId, which displays the variables deemed most important for affecting the significance of a model.

E. Comparing the Random Forests

Note that the code used for making the visualizations in this section is titled "ComparingModels.R".

After creating the random forest models with and without the turbine ID, we wanted to do a deeper analysis of not just how the models performed overall, but to also further see where the models' weak spots were. We primarily examined how they performed across two different variables that they were not trained on: FaultCodeType, and TimeUntilFault.

First, we wanted to see how well the models performed on each specific fault code type. We only used the 30% of the data that was not trained on for this analysis, which meant that the

sample size is somewhat limited for some of the fault code types. Figure 11 shows how well each model is able to correctly predict when there will be a fault code of each type. While for most of the fault code types the model is fairly accurate, the fault code types of pitch system and electrical are notable exceptions. While the random forest model with the turbine ID is able to do fairly well for electrical faults, the model without turbine ID performs poorly, incorrectly predicting electrical faults more than half of the time. Both models also have difficulty predicting pitch system faults; it is one of the few instances where the model without turbine ID performs better than the model with it (as shown in Figure 11).

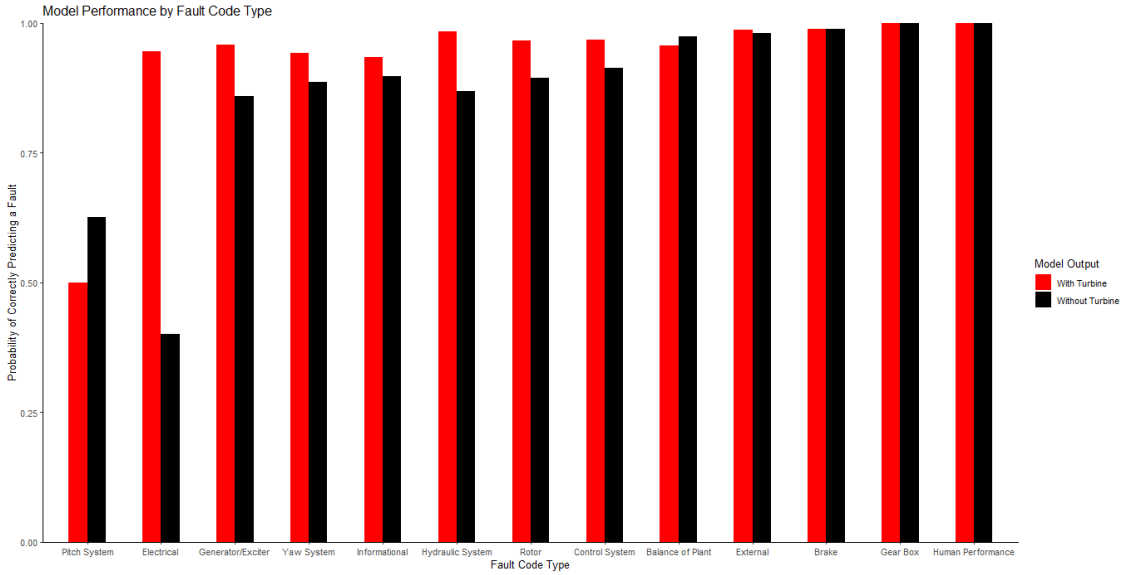

Figure 11: This graph shows how the models perform at correctly predicting a fault.

It is further important to note the way in which we are doing this analysis. We are not checking each fault code event and seeing if it was predicted correctly. Rather, we are looking at each row from the sensor readings data that had a fault occur within the next 24 hours, as well as looking at the percentage of the time that those rows correctly predicted a fault code occurrence within the next 24 hours. For example, there are 9 pitch system fault events, with 22 rows of

sensor readings in the testing dataset associated with those 9 events. This graph shows that the model with turbine ID predicts half of the 22, not half of the 9. In an ideal world with more time, it would be valuable to check how many of the 9 we predicted. If we spent more time working on this project, it would be valuable to re-format the data to be able to recreate this by a fault code event by fault code event basis, as opposed to a sensor reading by sensor reading basis.

Then, we wanted to see how each of the models perform depending on how far away they are from a fault code event (refer to Figure 12). To do this, we rounded the TimeUntilFault to the nearest hour and created bins based on the time until the next fault event occurs. Afterwards, we calculated for each bin what percent of the time the model predicted a fault code within the next 24 hours. For example, we took all of the data that was about 14 hours away from the start of a fault code event (which, in the testing set, was 251 observations) and calculated that 90.4% of those observations predicted a fault. We then made a line graph showing how each model performs based on the time until the next fault code event occurs, portrayed in Figure 12.

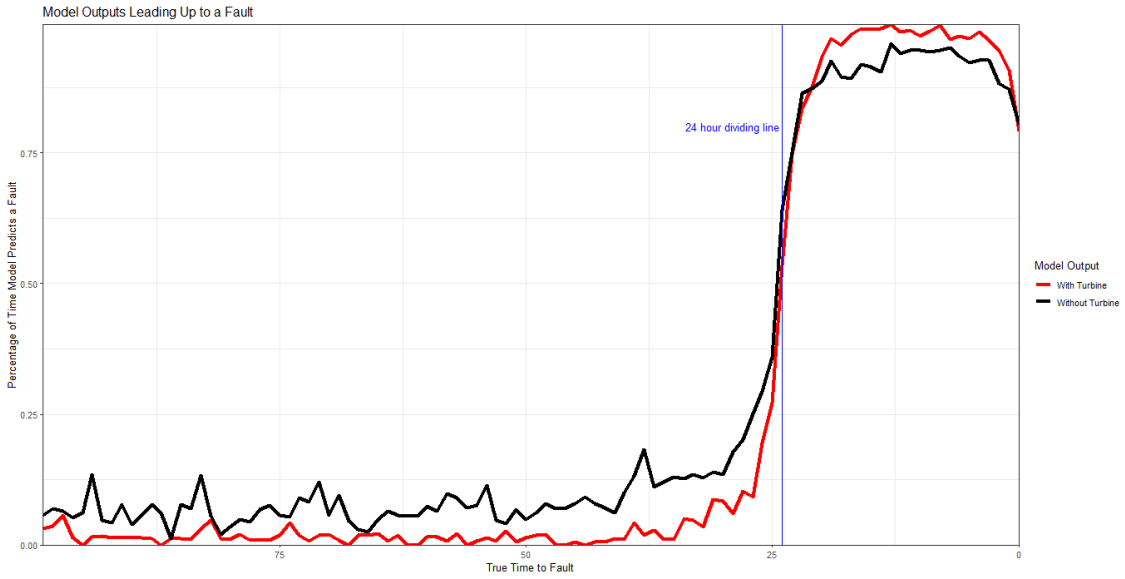

Figure 12: This graph shows what the models output depending on the time until the next actual fault code event occurs. The red line is the model with TurbineID, and the black line is the model without it. The blue line is the 24 hour cutoff where we consider a fault code event to start soon. We want the models to output a 0 leading up to the line, and a 1 after it.

There is a fairly steep change in the models' predictions at this 24 hour mark. The models are less accurate in the 22-26 area, which makes sense. Sensor readings are similar during those times, but the values for the target variable are different. Interestingly, the other time that the models' accuracy decreases is when a fault event starts within 2 hours. This goes against what we expected. Intuition would guess that the models perform really well when a fault is about to start, then get worse as the fault is further out. Instead, the models are worse trying to predict fault events starting soon. There was nothing particularly notable about how the models compare to each other. We know the model with the turbine ID performs better, and there does not seem to be a time until fault when the model without turbine ID notably outperforms the model with it.

V. Conclusion and Discussion

A. Limitations of the Models

We made a lot of somewhat arbitrary decisions over the course of this project that could have negatively impacted the performance of the models. The first one that we made was rounding to 30 minute intervals. It is possible that 30 minutes is too long, and we lose too much data. It is also possible that 30 minutes is too short. For example, if values are fluctuating crazily from the end of one 30 minute window to the start of the next, that is likely concerning and should be noted by the model. However, with 30 minute time intervals, this isn't possible.

Another arbitrary decision that we made was in the creation of fault events. We decided that, if two identical faults are within 6 hours of each other, they are part of the same fault event. Without more knowledge of the faults and the turbines on a more mechanical level, it is difficult to know where that line should be set. We also filtered some of these faults out if we deemed them to be irrelevant. It is possible that we filtered out some faults that are very relevant, or that there are faults that we left in that are unimportant and confusing the model.

When creating our final dataset, we made the decision to set the cutoff at 24 hours when determining whether a fault code occurs soon or not. Changing this 24 hour mark is partially a business decision (do you want to shut off a turbine if it is almost a full day away from faulting) and partially a data decision (do models make accurate predictions when the cutoff is increased).

Additionally, with regards to the weather data, we made multiple assumptions. We used Des Moines as a proxy for the entire state of Iowa, which is not ideal. We also only have daily weather data, which is not nearly as granular as the rest of the data that we have.

There are also some limitations in the scope of our models. We only had data from June 2 through September 2, on Iowa turbines only. Because of this, especially with the weather data, it is difficult to use our models outside of Iowa turbines in the summer.

Lastly, while the random forest models tell us that variables like mean ambient temperature are important for predicting a fault, they do not tell us the direction of that trend. We do not know whether high or low values of that variable are more likely to lead to a fault.

B. Conclusion

To summarize, we set out to make models to have a better understanding of what causes a fault code event, as well as predict when a fault code event will occur. We created graphs to show how interactions between certain variables can show that there will be a fault code event

starting soon. We also created a pair of random forest models to predict whether a fault would occur, one of them trained using turbine ID and one trained without it. These models are able to perform decently well, but have some blind spots; they perform poorly when trying to predict pitch system and electrical faults, as well as faults that are going to start within a few hours. Going forward, there are a lot of opportunities to improve on these models. Almost all of the limitations that we listed would be realistic to investigate and overcome with more time, data, and expertise about turbine mechanics. We hope that the work that we have done is able to provide value to BHE, and that they are able to continue and improve upon this research.