Bosch Production Line Performance
Capstone 1 – Final Report
Tom Widdows

Table of Contents

Introduction

Problem statement

A good chocolate soufflé is decadent, delicious, and delicate. But, it's a challenge to prepare. When you pull a disappointingly deflated dessert out of the oven, you instinctively retrace your steps to identify at what point you went wrong. Bosch, one of the world's leading manufacturing companies, ensures that the recipes for the production of its advanced mechanical components are of the highest quality and safety standards. Part of doing so is closely monitoring its parts as they progress through the manufacturing processes. Because Bosch records data at every step along its assembly lines, they have the ability to apply advanced analytics to improve these manufacturing processes. However, the intricacies of the data and complexities of the production line pose problems for current analytical methods.

The goal is to predict internal failures using thousands of measurements and tests made for each component along the assembly line. This would enable Bosch to bring quality products at lower costs to the end user.

Figure 1 - Bosch production facility

Kaggle

Expanded problem statement

I expanded the goal of predicting internal failures to include locating the source(s) of the failure (when possible) with the hope of assisting with identifying and solving the root cause(s) of the problem(s).

Data Collection and Cleaning

Data source

The data source for this project was a past Kaggle competition. The data represents anonymized measurements of a part as it moves through production line(s). Each feature heading is coded to include the production line, the station on that line and a feature number. For example, feature column header 'L1_S17_F4548' would represent line 1, station 17 and feature 4548. The raw data has been separated into three comma delimited files representing numerical, date and categorical information. The project focuses on the numerical and data information.

High level data overview

The numerical and date information from the source files are summarized below (figure 2):

```
Product Data (rows)
   Training rows:                    1,183,747
   Testing rows:                     1,183,748
   Total rows:                                       2,367,495

Features (columns
   Numeric Features:                       968
   Date Features:                        1,156
   Total Features:                                       2,124

Various Statistics
   Failed Products (rows):               6,879
   Total Products (rows):            1,183,747
   Failure Rate:                                         1/172
   Failure Rate Percent:                                 0.58%


~Potential Data Elements (features x rows):          5,028,559,380
```

Figure 2 - Data Overview

Data structure and initial preparation

The dataset has over 2,000 anonymized features. Because of the size and number of data elements, the dataset required reshaping for cleaning and Exploratory Data Analysis (EDA). I initially split the numeric and date CSV files into 10 equally sized files so I had the memory and processing power to read and manipulate the data. I utilized Pandas melt to reshaped the data in each file by transforming the column headers into rows ultimately making the tables significantly narrower and longer. During this process, I added three columns named Line, Station and Feature and populated the columns with data from the column header. For example, a column header of 'L1_S17_F4548' would result in the Line, Station and Feature columns being populated with 1, 17 and 4548 respectively. Finally, I concatenated the reshaped data and stored the results as new CSV files. This new format made it possible to analyze the data much easier by line and station.

Id	L0_S0_F0	L0_S0_F2	L0_S0_F4
4	0.030	-0.034	-0.197
6	NaN	NaN	NaN
7	0.088	0.086	0.003
9	-0.036	-0.064	0.294
11	-0.055	-0.086	0.294

Figure 3 - Original source data

Id	Old_Name	Line	Station	Feature	Value
4	L0_S0_F0	0	0	0	0.030
4	L0_S0_F2	0	0	2	-0.340
4	L0_S0_F4	0	0	4	-0.197
6	L0_S0_F0	0	0	0	NaN
6	L0_S0_F2	0	0	2	NaN
6	L0_S0_F4	0	0	4	NaN
7	L0_S0_F0	0	0	0	0.088
7	L0_S0_F2	0	0	2	0.086
7	L0_S0_F4	0	0	4	0.003

Figure 4 - Wide to long modified source data

Although this approach worked most of the time (occasionally Pandas Melt would fail), it was slow and required significant RAM. Ultimately, I created 3 additional data frames for line, station and pod by querying the column headers in the main file to get the subset of data needed for each new column. This proved to be much more efficient.

The numeric and date data consists of approximately 80% missing values. Since the values of the measurements are normalized, I ultimately plan to fill any missing values with zero.

Because of having anonymized features and the volume of data, I did not detect any outliers. In addition, because of the raw volume of data, it is unlikely any outliers (unless extreme) would impact the ultimate model performance.

Exploratory Data Analysis

Train vs test

Let's begin by comparing our training and test sets to confirm the volume of product through each line and station is similar (since the datasets for each are effectively the same size).

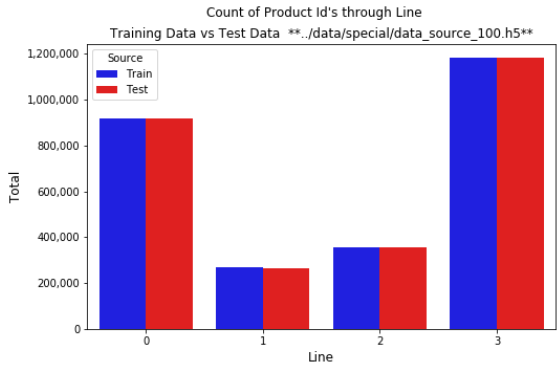


Figure 5 - Train and test volume through lines

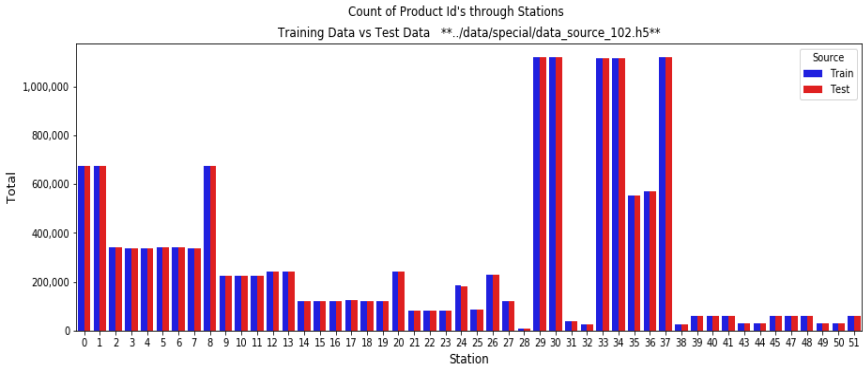


Figure 6 - Train and test volume through stations

The test and train data mirror each other with respect to the volume of product that travels through each line, and within each line, each station.

Construction of test and training files

With some inspection, it is very plausible Bosch produced 2,367,495 individual products manufactured over an undefined period (more on that soon), shuffled the order, gave each product a new id starting at id = 1 and randomly selected 1/2 of the parts for the test dataset. Figure 7 is from the training set and figure 8 is from the test set. If the two datasets were combined, the Id numbers would be continuous. This generally holds true throughout the entire files.

Training Dataset (with outcomes)

Id	L0_S0_F0	L0_S0_F2	L0_S0_F4
4	0.030	-0.034	-0.197
6	NaN	NaN	NaN
7	0.088	0.086	0.003
9	-0.036	-0.064	0.294
11	-0.055	-0.086	0.294

Figure 7 - Training dataset header

Testing Dataset (without outcomes)

Id	L0_S0_F0	L0_S0_F2	L0_S0_F4
1	NaN	NaN	NaN
2	NaN	NaN	NaN
3	NaN	NaN	NaN
5	-0.016	-0.026	-0.033
8	NaN	NaN	NaN

Figure 8 - Testing dataset header

Station 32

When reviewing the error rate by station, it is clear something different is happening at station 32. 4% of that parts that go through station 32 are defective. This is ~8 times the average failure rate. Another interesting fact about station 32 is it is the only station that has only one feature (process). This will certainly be one area of focus as we develop the prediction model.

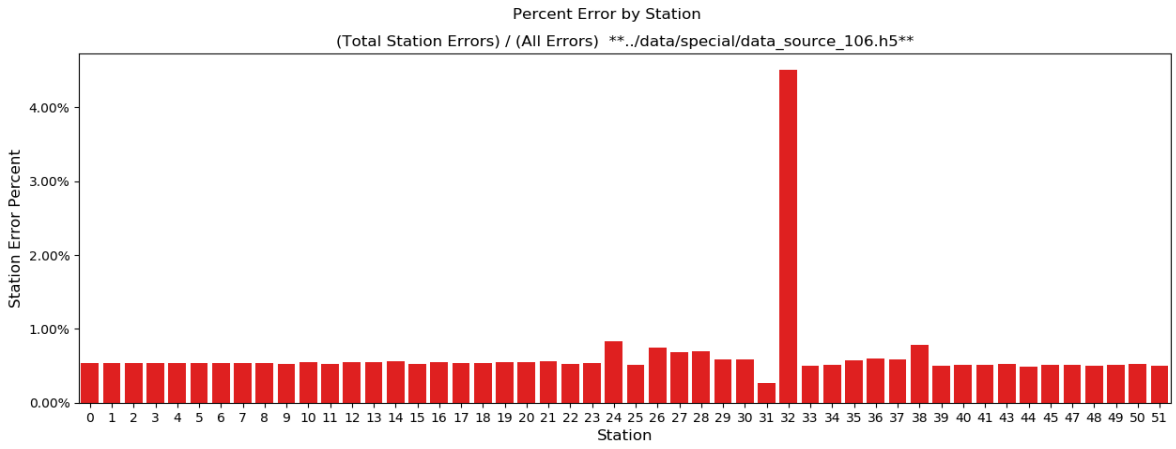


Figure 9 - Station 32 error rate

Consecutive errors

Figure 8 below shows 345 consecutive failures (based on product id), much higher than the expected value of 40 if the failures were random and a sample was not extracted from the dataset. The occurrence of consecutive failures on id was puzzling because the production rows appear to have been randomized before assigning consecutive ids. Upon further inspection, when product id's were sorted in date order, the volume of failures in consecutive products produced was still statistically significant. Hypothetically, if a raw material purchased was defective, it would create a failure. If that raw material was used in two parts, both parts would be defective. An example might be a small sheet of aluminum that is cut in half and used in two parts. It remain unclear if this would be useful in a production environment.

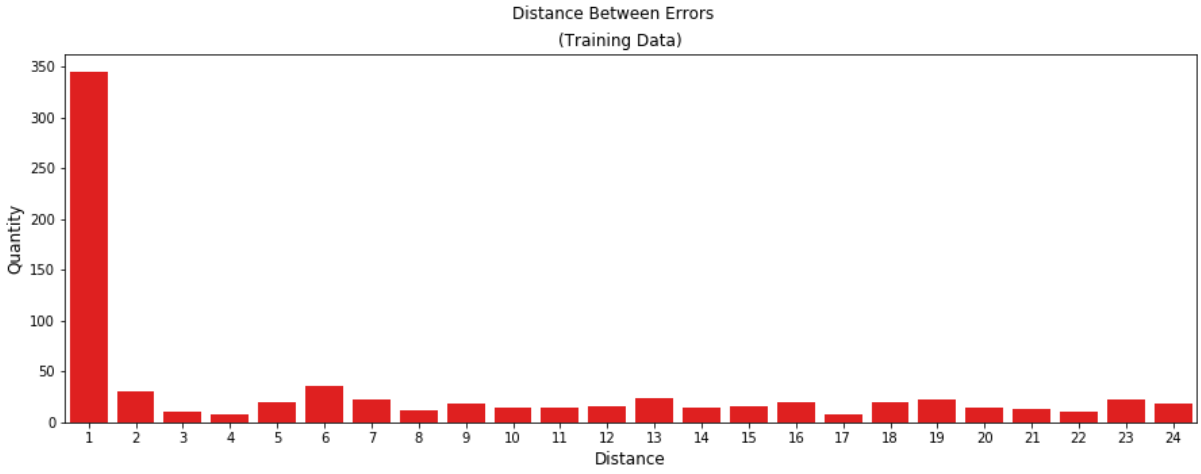


Figure 10 - Consecutive failures

Part routes

For the graph below, a part route is determined by which lines a part goes through during manufacturing. For example, all parts that go through only line 2 and line 3 would have the same part route id. The desire is we can find some correlation between what route a part takes and defective rates. In addition to line routes, we can calculate station routes. My initial belief is line routes summarize the data to the point we cannot find any significant correlation to failures, but station routes (and pod routes discussed below) have a better chance of showing a useful correlation to failures.

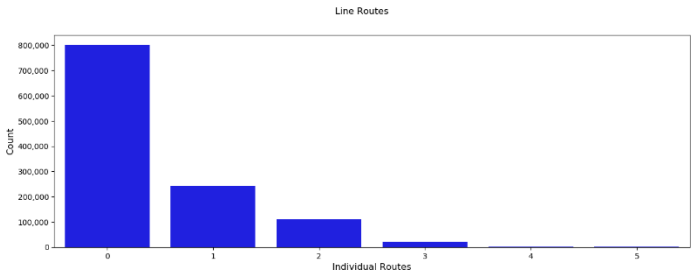


Figure 11 - Line routes

Creation of Pods

As a product travels through production, it travels through 1 or more lines as well as multiple stations (which are all contained within a line). Reviewing Figure 12 below, a part would travel left to right through the different lines ultimately becoming a finished good. The production volume in Figure 12 is stored by station 0, 1 … 51. It is relatively easy to spot all kinds of patterns in the figure. For example, a if a product goes through station 0, it will allways go through station 1 and then it will go through station 2 or 3 but never both. Another pattern easy to spot is how parts that start at station 0 will always go through statsions 9, 10 or 11 but only one of them. Figure 13 shows the like stations combined and referred to as Pods.

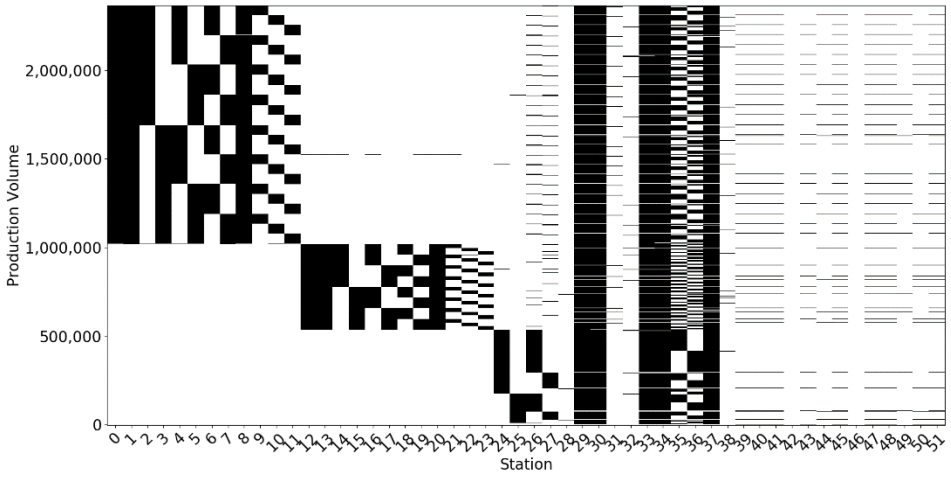


Figure 12 - Production through stations

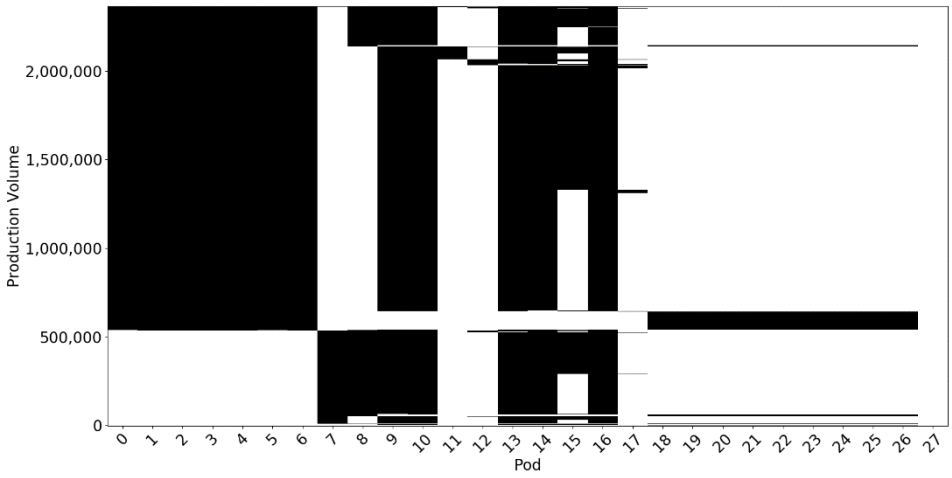


Figure 13 - Production through pods

Measurement dates

The date measurements were recorded as a two-decimal number, for example 157.14. After extracting and sorting the date features, gaps could be identified in the measurements and it became easy with a little experimentation to identify weekends and ultimately conclude that each .01 represents 6 minutes. With a little more experimentation, it became clear (at least highly likely) the times represented January 1st, 2014 through December 31st, 2015. This provided an opportunity to engineer addition features to our data like production day of week, time of day, month of year… Figure 13 shows production volume for the first quarter of 2014 with weekends highlighted in light brown.

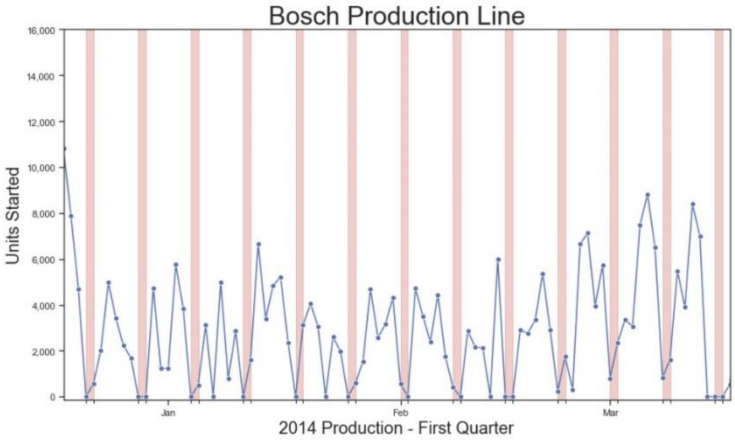


Figure 14 - Measurement dates

Que and Work in Process (WIP)

For each line, station and pod I calculated the volume of product waiting to be processed (que) and the volume of product in work-in-process (WIP). As noted in the future work below, I neglected to calculate production through the location, but it would be relatively easy to calculate. With that information, production volume through the location could be calculated for any time period (e.g. last hour, last 30 minutes…)

Technical Approach

Format of data to analyze

The data (including engineered features) is in 4 separate data frames as follows:

- Original data (all detail)
- Station (a subset of original data)
- Pod (a subset of Station)
- Line (a subset of Pod)

My approach is to combine the best predictive features from each of the 4 areas and create a set of features to utilize for feature selection in the final model.

Model selection

The goal of this project is to predict product failures using the provided data. This is a classification problem as the outcomes are good (pass) or bad (fail). Below is a summary of the machine learning models evaluated:

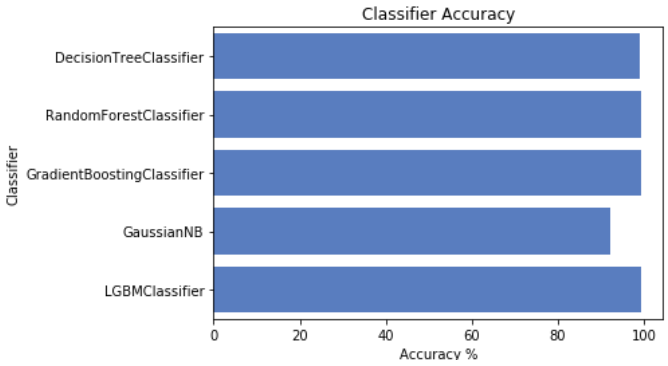


Figure 15 – Classifier accuracy

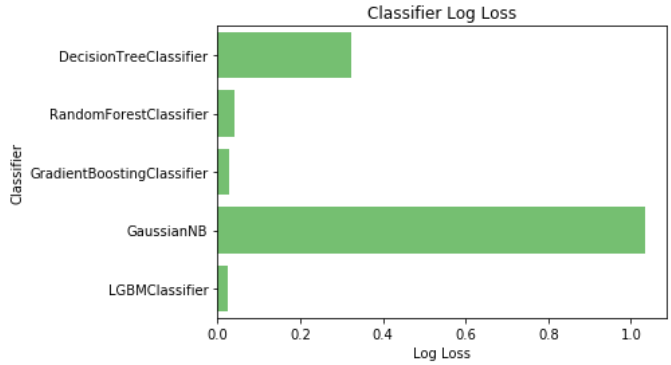


Figure 16 – Classifier log loss

Comparing the various models, LightGBMClassifier (LGBM) had the lowest Log Loss. LGBM, RandomForestClassifier (RF) and DecesionTreeClassifier (DT) had very similar accuracy scores but LightGBM ran in under 10 seconds where RF and DT took far longer. I opted to train the data using LightGBM.

Model Evaluation

The client dictated we utilize the Matthews Correlation Coefficient (MCC) for model evaluation.

$$\mathrm{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Figure 17 – Matthews Correlation Coefficient (MCC)

Where:

- TP = true positive (we predicted positive and it was!)
- TN = true negative (we predicted negative and it was!)
- FP = false positive (we predicted positive, but it was actually negative)
- FN = false negative (we predicted negative, but it was actually positive)

Reference Model

A reference model was run on the original data with no engineered features. Generic / default parameters were used when training the data.

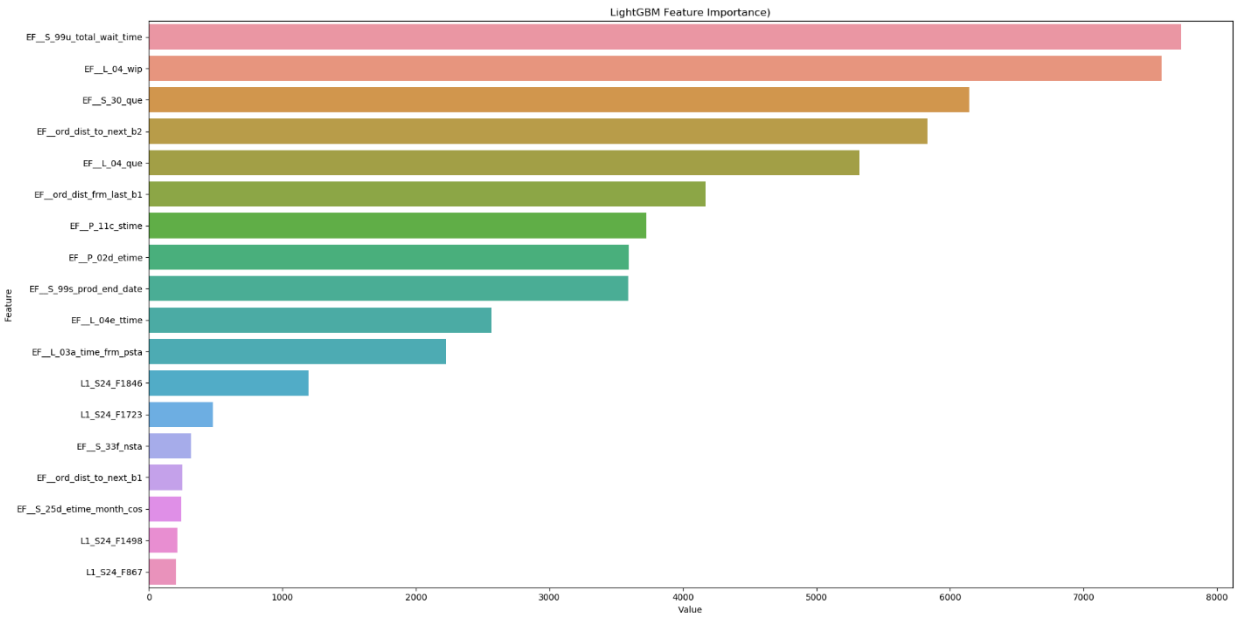
Feature selection and elimination

I utilized SelectFromModel to select any predictive features from the 4 data frames as discussed above. Those features were combined and run through the LGBMModel as discussed below.

The LGBMClassifier gave the lowest log loss, very good accuracy and executed in under 10 seconds. As described above, the feature selection utilized LGBMModel which is the parent class of LGBMClassifier. Removing any of the features in the plot below has a negative impact on the model score while adding any addition features does not improve the score and often has a negative impact on the score.

All features starting with "EF__" are engineered features:

- EF__L = Line engineered feature
- EF__S = Station engineered feature
- EF__P = Pod engineered feature
- EF__ord = Engineered feature related to production sequence



Hyperparameter optimization

Parameter optimization, often referred to as tuning, is the process of selecting the optimal parameters for a learning algorithm. Hyperopt was used for parameter optimization. Hyperopt ran 500 training simulations to locate the optimal values for 7 parameters: n_estimators, learning_rate, num_leaves, subsample, colsample_bytree, reg_alpha and reg_lambda.

Final Results

The reference model consists of training on the provided data and yielded a Matthew Correlation Coefficient (MCC) 21.58%

After feature engineering, feature reduction and some changes to the hyperparameters, the model provided a MCC of 41.45%

After hyperparameter optimization, the model yielded a MCC of 42.50%

Finally, the final model with optimized hyperparameters against Hold-Out/Test data and received a MCC of 44.88%

Client Impact

By utilizing the data already being collected during manufacturing, the trained model was able to identify 17% of the defective product (1,100+) with only 35 good products being misclassified as bad products. Since we have proven we can detect failed product with the trained model, we should replace MCC as the evaluation metric in favor a custom metric that minimizes the total cost to Bosch. To accomplish this, we would need the cost to Bosch of a failed product that was not detected and the cost of a good product being incorrectly classified as bad. With this information, we could use the same model but optimize on minimizing the total cost to Bosch.

Finally, it is important to utilize the predictive features in the model to locate and correct the root cause of the failures.

Future Work

- Calculate production volume for lines, stations and pods (could be incorporated with calculating que and WIP easily)
- Improve feature selection from the 4 data sources including looking at covariance between features.
- Minimize the total cost to Bosch by determining the cost of a failed product being incorrectly classified as good and the cost of a good product being incorrectly classified as bad. With this, we could optimize the model for the maximum positive financial impact to Bosch.