Capstone Project:

Bosch Production Line Performance

Machine Learning Engineer Nanodegree

Joseph Durago

12/18/2016

I. Definition

Project Overview

Bosch, a company which manufactures automotive components, industrial products and building products, shared production line data with Kaggle. The data consists of measurements from 4 production lines and 50 stations. The measurements follow a part as it goes the the production line and is a mixture of numerical, categorical, and time data. Bosch and Kaggle have started a data science competition in hopes to predict whether a part will fail or pass Bosch's quality control. The data set can be found on Kaggle's website:

https://www.kaggle.com/c/bosch-production-line-performance/data

Developing a model of passing or failing parts is important for any manufacturer. The methods developed from this project can be used to identify weak points in production lines that contribute to a part failing and hopefully to correct them. Also, stations or equipment that contribute to failing parts can be monitored more closely. If the measurements from this station are out-of-spec then the part could be removed before it goes further down the production line.

Problem Statement

The goal of this project is to accurately predict if a part will pass or fail Bosch's quality control using the training and test data provided by Bosch. I will be implementing an XGBoost (Extreme Gradient Boosting) algorithm, which is a supervised learning technique based on decision trees. I chose to use the XGBoost algorithm as it has been used in several winning Kaggle competitions. Also, XGBoost has comparatively good runtime performance versus other machine learning algorithms. This is important since the data set is quite large. To implement the XGBoost algorithm I will need to do the following:

1. Identify machine that can process the large dataset
2. Explore the dataset to determine important and unimportant features
3. Implement XGBoost algorithm based on important features
4. Tune algorithm to maximise correct predictions of passing/failing parts

Important and unimportant features will be identified by taking a sample of the dataset. The XGBoost algorithm will be fitted to this sampled dataset. A method within XGBoost allows important features to be identified. After these important features are identified I'll load all the data, but only from important features that were previously identified. Once all the data has been loaded I'll train and tune XGBoost to achieve the best prediction score.

Metrics

For the Kaggle Bosch Production Line Performance competition, all submissions are evaluated on the Matthews correlation coefficient (MCC) between the predicted and the observed response. The MCC is given by:

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

where TP is the number of true positives, TN the number of true negatives, FP the number of false positives, and FN the number of false negatives.

MCC returns a value between -1 and +1. A coefficient of +1 represents a perfect prediction, 0 no better than random prediction and −1 indicates total disagreement between prediction and observation. [1] MCC is a good metric in assessing the accuracy

of the production line data as it takes into account true positives, true negatives, false positives and false negatives in determining model performance.

F1 score is another metric that could have been used for evaluation. It is defined as [2]:

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Where precision and recall are defined as [3]:

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

Comparing the MCC and F1 equations, we can see that F1 does not take into account true negatives. Additionally, MCC performs well with unbalanced data sets or data sets where there is a high frequency of one value versus another [4] . Since the Bosch production line data is composed mainly of passing parts, MCC is a good metric for evaluating the performance of the model. For these reasons I'll be using the MCC as the main metric for determining performance of XGBoost. However, I will use F1 score in some data visualizations. This is because the XGBoost python module has built-in charting functions that utilize the F1 score.

II. Analysis

Data Exploration

The dataset consists of 7 files and is a mix of categorical, numerical, and date data:

Filename	# Rows	# Columns
test_categorical.csv	1183749	2141
train_categorical.csv	1183748	2141
test_date.csv	1183749	1157
train_date.csv	1183748	1157

test_numeric.csv	1183749	969
train_numeric.csv	1183748	970
sample_submission.csv	1183749	2

https://www.kaggle.com/c/bosch-production-line-performance/data

The dataset is 17 GB. Working with a large dataset is a challenge, especially one with a sizeable number of columns. I'll be using a r3.xlarge instance on Amazon Web Services, which has 30.5GB of memory and 160GB of SSD storage to aid in data processing.

The dataset contains an extremely large number of features that have been anonymized (likely for intellectual property reasons). Features are named according to a convention that tells the production line, the station on the line, and a feature number. E.g. L3_S36_F3939 is a feature measured on line 3, station 36, and is feature number 3939. The date features provide a timestamp for when each measurement was taken. Each date column ends in a number that corresponds to the previous feature number. E.g. the value of L0_S0_D1 is the time at which L0_S0_F0 was taken.[5]

The categorical data set is primarily composed of single value data points. Fig 1 the distribution between no entries, single value entries, and multi-value entries.

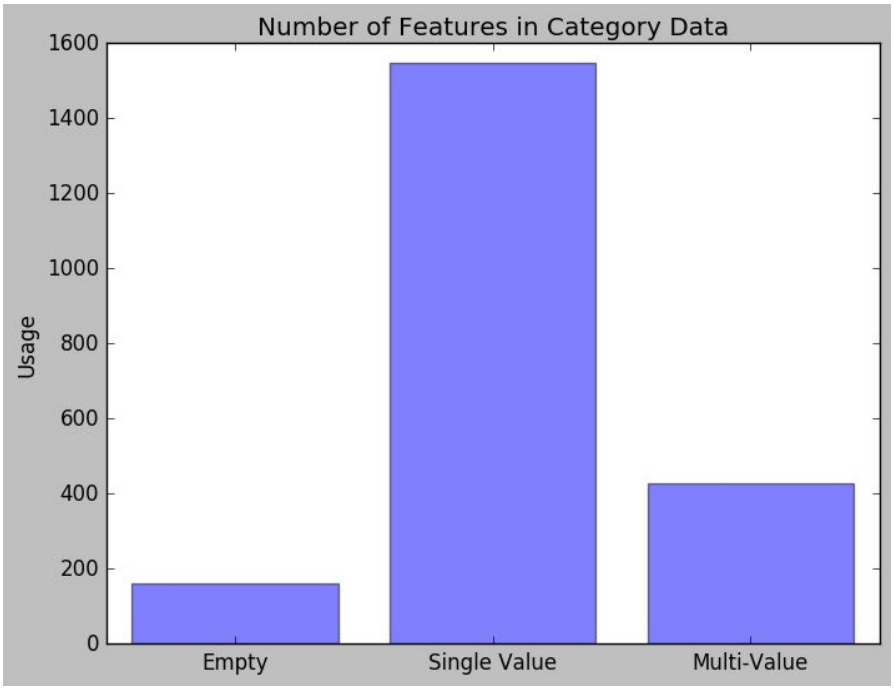


Fig 1.

XGBoost requires categorical data to be encoded for proper classification. However, with such a large data set, one hot encoding all of the categorical data from Bosch will make the data set very large and difficult to compute on. To save space, I plan on using an encoding technique called, leave-one-out encoding [6].

Fig 2. shows there are significantly more passing units (Fail=0) as opposed to failing units (Fail=1) in the training data. There are 1176868 passing parts and 6879 failing parts. This is a 0.58% failure rate and seems like a reasonable number for a high volume manufacturer.

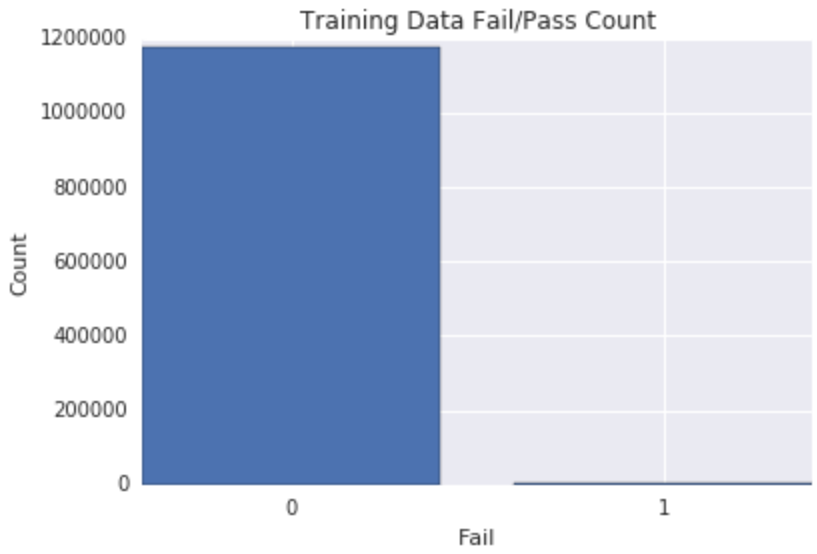


Fig 2

Below is a small sample of the categorical, numerical and date training data along with some summary statistics of 4 columns from each data type.

Numeric

Descriptive Statistics

	L0_S0_F2	**L0_S0_F4**	**L0_S0_F6**	**L0_S0_F8**
mean	0.000091	0.00004	0.000014	-0.000126
std	0.092799	0.21152	0.211635	0.094555
min	-0.616	-0.415	-0.416	-0.447
25%	-0.056	-0.179	-0.179	-0.056

50%	0.004	-0.033	-0.034	0.031
75%	0.063	0.294	0.294	0.074
max	0.302	0.585	0.584	0.466

Sample of Raw Data

Id	L0_S0_F2	L0_S0_F4	L0_S0_F6	L0_S0_F8	Response
4	-0.034	-0.197	-0.179	0.118	0
6	NaN	NaN	NaN	NaN	0
7	0.086	0.003	-0.052	0.161	0
9	-0.064	0.294	0.330	0.074	0
11	-0.086	0.294	0.330	0.118	0
13	0.019	0.294	0.312	0.031	0
14	NaN	NaN	NaN	NaN	0
16	NaN	NaN	NaN	NaN	0
18	-0.041	-0.179	-0.179	-0.056	0

Categorical

Sample of Raw Data

Id	L3_S35_F3885	L3_S35_F3887	L3_S35_F3888	L3_S47_F4142
212468	T1	T1	T16	NaN
478899	T1	T1	T16	NaN
721551	T1	T1	T16	NaN
742342	T1	T1	T16	NaN
947649	T1	T1	T16	NaN
1352021	T1	T1	T16	NaN
1501926	T1	T1	T16	NaN
1689886	T1	T1	T16	NaN
1810776	T1	T1	T16	NaN

2204849	T1	T1	T16	NaN

Date

Descriptive Statistics

	L0_S0_D3	L0_S0_D5	L0_S0_D7	L0_S0_D9
mean	882.229068	882.229068	882.229068	882.229068
std	506.724916	506.724916	506.724916	506.724916
min	0.000000	0.000000	0.000000	0.000000
25%	392.800000	392.800000	392.800000	392.800000
50%	909.230000	909.230000	909.230000	909.230000
75%	1374.450000	1374.450000	1374.450000	1374.450000
max	1713.710000	1713.710000	1713.710000	1713.710000

Sample of Raw Data

Id	L0_S0_D3	L0_S0_D5	L0_S0_D7	L0_S0_D9
4	82.24	82.24	82.24	82.24
6	NaN	NaN	NaN	NaN
7	1618.70	1618.70	1618.70	1618.70
9	1149.20	1149.20	1149.20	1149.20
11	602.64	602.64	602.64	602.64
13	1331.66	1331.66	1331.66	1331.66
14	NaN	NaN	NaN	NaN
16	NaN	NaN	NaN	NaN
18	517.64	517.64	517.64	517.64

Exploratory Visualization

Fig 3 compares the error rate of each of the 4 production lines. The error rates between the production lines are relatively similar.

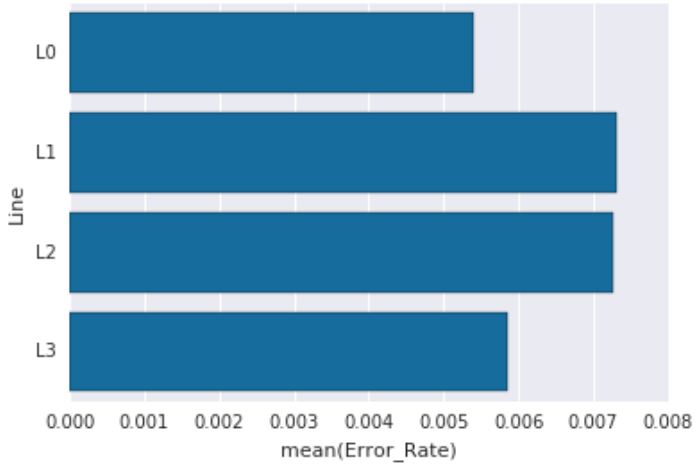


Fig 3

Fig 4 compares the error rates of the various stations. Station 32 has a much higher error rate compared to the other stations. I would speculate that the station could be a rework station where faulty components go to be refinished. Given the high error rate associated with this station, I would expect the XGBoost algorithm to incorporate it in its model for predicting failed components.

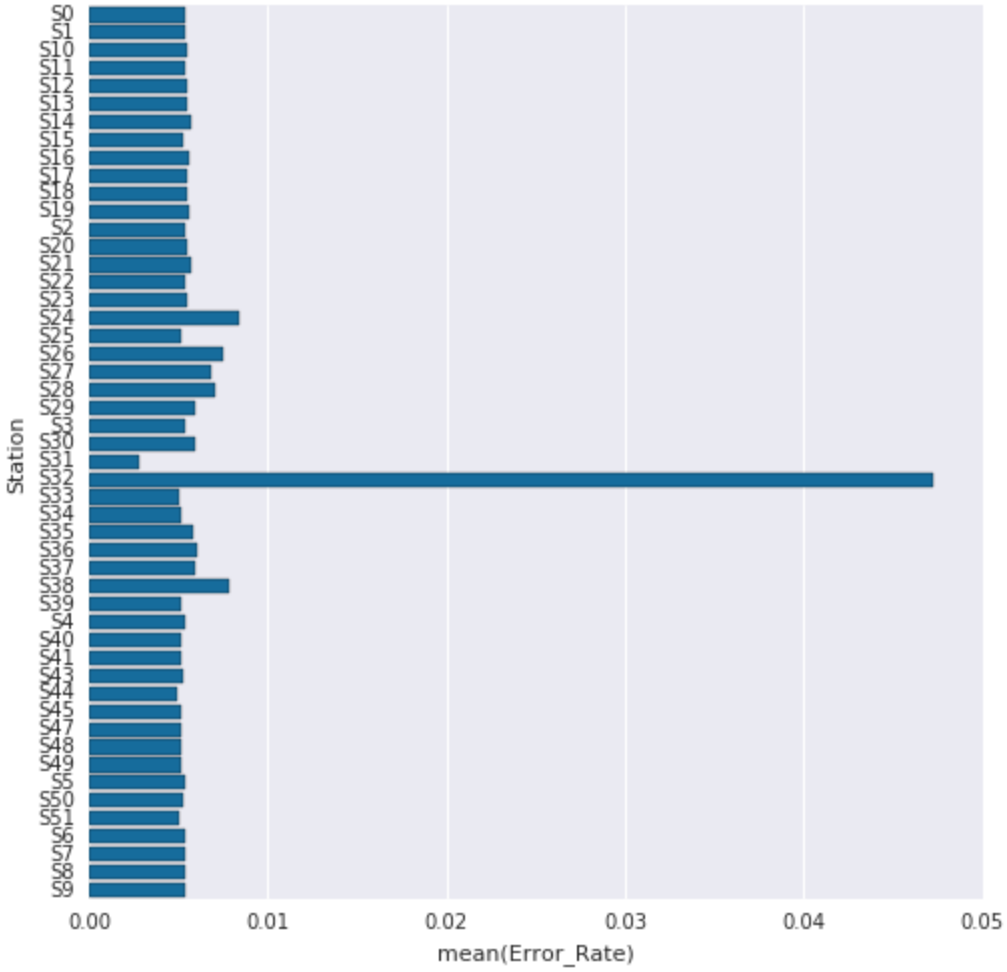


Fig 4

Algorithms and Techniques

XGBoost

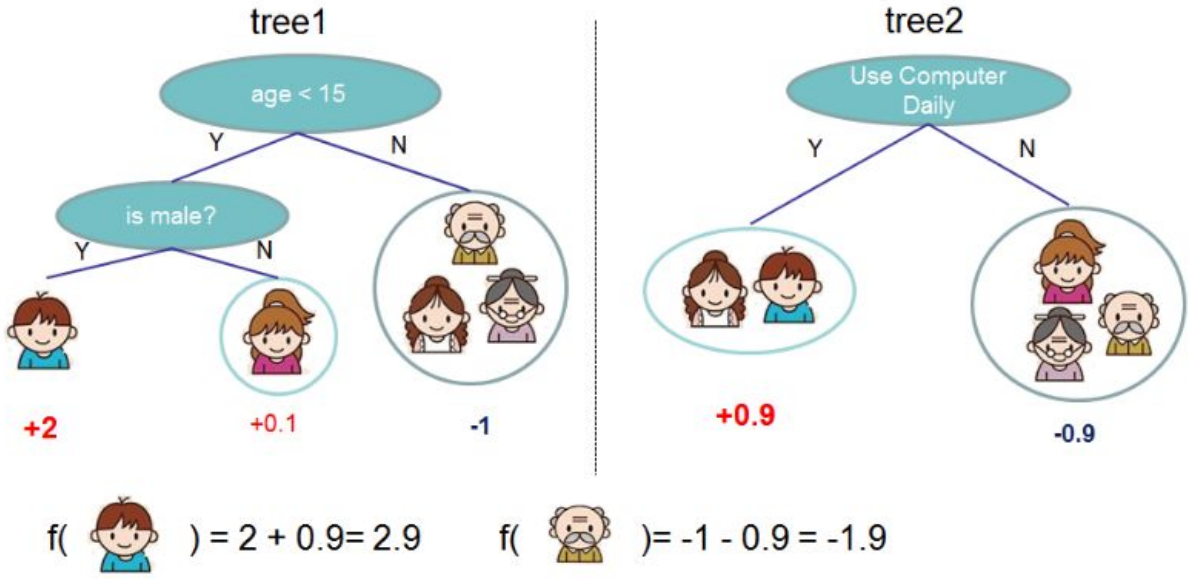
An XGBoost algorithm will be used in developing a model from the training data. The algorithm is based off of a decision tree. There are various decision trees algorithms and each apply different metrics to form the tree. For example the ID3 algorithm attempts to minimize entropy, or the amount of uncertainty in a data set, while maximizing information gain. XGBoost is based on an ensemble of trees and its objective is to minimize training loss and place higher value on simpler models.

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Training loss Complexity of the Trees

[4]

XGBoost is an ensemble of trees. During training, a new tree is added with each prediction. Each of the trees attempts to optimize the objective function above. Given an input a prediction is made by taking the weights of each of the trees in the ensemble. Individually each tree is a weak classifier. However, the ensemble of trees together create a strong classifier.

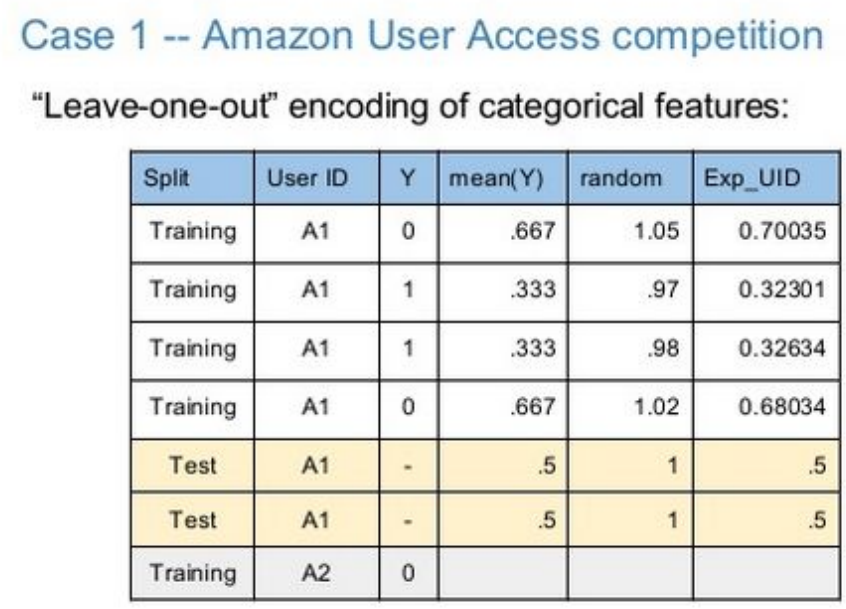


The benefits of using an XGBoost algorithm is that it is invariant to inputs so normalization is not needed, high order interactions can be learned between features, its scalable, and is used widely in industry. Together, these benefits make XGBoost an ideal candidate to use for the Bosch production line data set. [7]

Leave-One-Out Categorical Data Encoding

The categorical data set is large. It is composed of 1183747 columns and the majority of columns only have a single-value entry. A column with a single-value entry either has a value of NaN or a value like ‘T1’. The ‘Leave-One-Out’ technique will be used to encode

the categorical data, so that XGBoost can be properly read and modeled. I'll be using this technique because it reduces the amount of columns and it has been used in winning Kaggle competitions [8]. Other feature selections such as Principal Component Analysis (PCA) or Recursive Feature Elimination (RFE) could be used, however the categorical data would still need to be encoded. Since the categorical data already has over a million columns, encoding the data would, at least, double the number of columns. The Leave-One-Out method allows the total number of columns to remain the same and encodes the categorical values so that they can be used by XGBoost.



Case 1 -- Amazon User Access competition

"Leave-one-out" encoding of categorical features:

Split	User ID	Y	mean(Y)	random	Exp_UID
Training	A1	0	.667	1.05	0.70035
Training	A1	1	.333	.97	0.32301
Training	A1	1	.333	.98	0.32634
Training	A1	0	.667	1.02	0.68034
Test	A1	-	.5	1	.5
Test	A1	-	.5	1	.5
Training	A2	0			

[9]

The leave-one-out encoding method is:

1) Calculate User ID category "A1" average response (Y) which is 0.5
2) Use average response from step 1in Test dataset where "A1" is found
3) For the training data, leave out the row's response and calculate the mean of all responses where "A1" is found in other rows, that yields the mean(Y) of the first row on the slide. mean(1, 1, 0) = 0.66
4) Add a little bit of noise and we end up with the 0.70035 at the last column of the first row
5) Repeat steps 1-4 for remaining categorical data

Benchmark

As a benchmark, I plan on implementing the XGBoost algorithm with the default parameters. The parameter values I plan on manipulating when tuning the algorithm are listed below [10]:

learning rate [default=0.3, alias: learning_rate]

- step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features. and eta actually shrinks the feature weights to make the boosting process more conservative.

max_depth [default=6]

- maximum depth of a tree, increase this value will make the model more complex / likely to be overfitting.
- range: [1,∞]

subsample [default=1]

- subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collected half of the data instances to grow trees and this will prevent overfitting.
- range: (0,1]

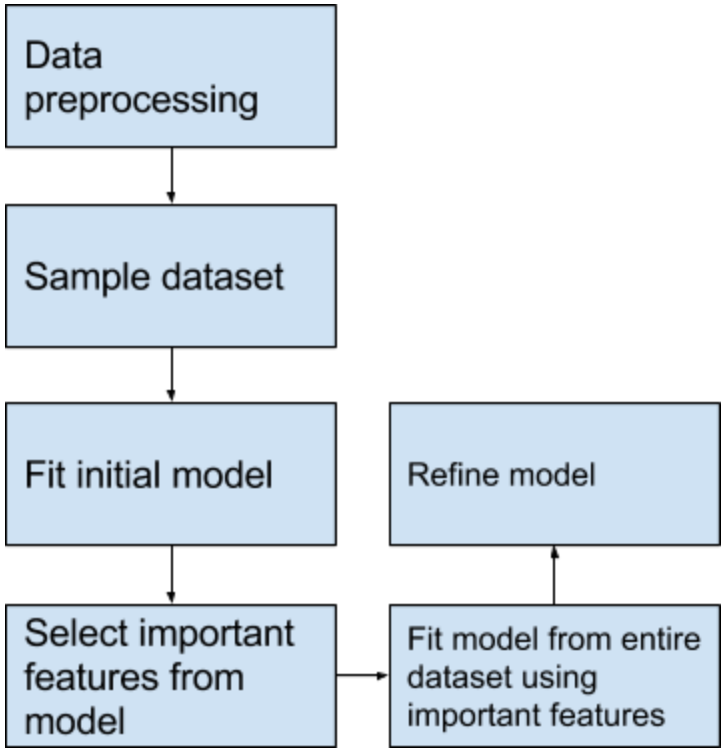
colsample_bytree [default=1]

- subsample ratio of columns when constructing each tree.
- range: (0,1]

With these default parameters I achieve an MCC score of 0.20118 and I would have been on the 891st place on the leaderboard. My goal is to tune the XGBoost algorithm to perform better than the benchmark.

III. Methodology

The flowchart below shows the methodology used in developing the XGBoost algorithm on the Bosch production line dataset and will be discussed further in subsequent sections.:



Data Preprocessing

The baseline algorithm with default parameters discussed in the Benchmark section provides an MCC score of 0.209. Incorporating all of the data and encoding all the categorical data would require lots of processing power. To reduce the amount of processing power needed, I filtered out insignificant features and used a leave-one-out encoding technique. I had also processed the date data into weekly and daily bins. This way the XGBoost algorithm could potentially find pass/fail patterns that could occur on a particular day of the week or on a particular time of day.

Implementation

After the data was preprocessed, I sampled 5% of the data and fit the XGBoost classifier to it using the following benchmark parameter values:

Learning rate = 0.3
Max_depth =6
Subsample = 1
Colsample_bytree = 1

Sampling the data was used so that I could quickly identify important features. The full dataset was then filtered to include only the previously identified important features and a XGBoost classifier was then fitted to the new data set. Sampling the data allowed for a reduced amount of processing power. I initially attempted to fit the entire data set and not sample the data. This was unmanageable, took lots of processing time and I used up all the available 244 GB of memory on my r3.8xlarge machine [11].

To show the high level functionality of the program, I have copied the main function below:

```
if __name__ == '__main__':
  print('Started')

  if PRE_PROCESS_CATEGORICAL_DATA:
    print('Started Pre Processing Categorical Data')
    pre_process_categorical_data()
    print('Finished Pre Processing Categorical Data')

  if PRE_PROCESS_DATE_DATA:
    print('Started Pre Processing Date Data')
    pre_process_date_data()
    print('Finished Pre Processing Date Data')

  print('Started important_cols')
  important_cols = identify_important_features(train_date_1wk_filename,
train_numerical_filename)
  print('Finished important_cols')

  print('Started loading train data')
  train, response = load_train_data(important_cols, train_date_1wk_filename,
train_numerical_filename)
  print('Finished loading train data')

  param = {}
  param['learning_rate'] = 0.3
  param['max_depth'] = 6
  param["subsample"] = 1
  param['colsample_bytree'] = 1
  param['base_score'] = 0.005
  clf = XGBClassifier(**param)

  clf, best_threshold = train_data(clf, train, response)
  print('Finished training')
  test_data = load_test_data(important_cols, test_date_1wk_filename, test_numerical_filename)
  predict_and_submit(test_data, best_threshold, clf)

  print('Finished')
```

Refinement

After preprocessing the data and training the XGBoost algorithm. I identified the important features that should be used in creating a model. Initially I trained on all the data, however I found that none of the categorical features were important, when using the leave-one-out encoding technique. Also incorporating the processed date data, which binned the data into a rolling weekly format didn't provide a more accurate prediction. However, using a rolling daily format for the date data seemed to make some improvement. To save processing time I did not use any of the categorical data on subsequent rounds of training and I relied on the rolling 24 hour date data instead of the raw date data and rolling weekly date data.

IV. Results

Model Evaluation and Validation

I tuned the XGBoost algorithm using a grid search approach. The parameters I tuned with the various values were:

learning_rate: [0.05, 0.1, 0.3]
max_depth: [6, 9, 12]
subsample: [0.9, 1.0]
colsample_bytree: [0.9, 1.0]

The final model had the final values:

Learning_rate = 0.3
Max_depth = 12
Subsample = 0.9
Colsample_bytree = 0.9

See table below for the parameter combinations and result:

#	Learning Rate	Max Depth	Subsample	Colsample By Tree	MCC Score
1 (Baseline)	0.3	6	1	1	0.201
2	0.05	6	0.9	0.9	0.219

3	0.1	6	0.9	0.9	0.202
4	0.3	6	0.9	0.9	0.199
5	0.05	9	0.9	0.9	0.214
6	0.1	9	0.9	0.9	0.202
7	0.3	9	0.9	0.9	0.214
8	0.05	12	0.9	0.9	0.203
9	0.1	12	0.9	0.9	0.205
10 (Final)	0.3	12	0.9	0.9	0.229
11	0.05	9	1.0	0.9	0.210
12	0.05	9	1.0	1.0	0.210

I achieved a MCC score of 0.22903 and I would have been on the 720th place on the leaderboard. This is an improvement of 171 places on the leaderboard from the benchmark model.

Comparing values from the table above, as learning rate increases the MCC score in general increases. XGBoost is able to reach the optimum point more quickly as the learning rate increases. Also, as max depth increase the MCC score in general increases. A larger max depth allows XGBoost to capture more complexities and nuances in the data, however it increases the likelihood of overfitting. Changing the subsample and colsample by tree seems to have little effect on the MCC score. Subsample is the fraction of observations to be random samples for each tree. Lower values make the algorithm more conservative and prevents overfitting but too small values might lead to under-fitting. A possible explanation for subsample not having significant effect is that there isn't much variation in the data. Thus including more or less observations that are random doesn't have much effect. Colsample by tree is the fraction of columns to be randomly samples for each tree. Since there are several million columns in the dataset I think changing this value by 10% doesn't have much effect on the final outcome.

Justification

Compared to the MCC benchmark score of 0.20118, I had achieved a score of 0.22903 after tuning the algorithm. This was a marginal improvement from the benchmark. I think the model I had developed is a good starting point, however there is much improvement to be made before my machine learning model should be deployed in a production setting.

V. Conclusion

Free-Form Visualization

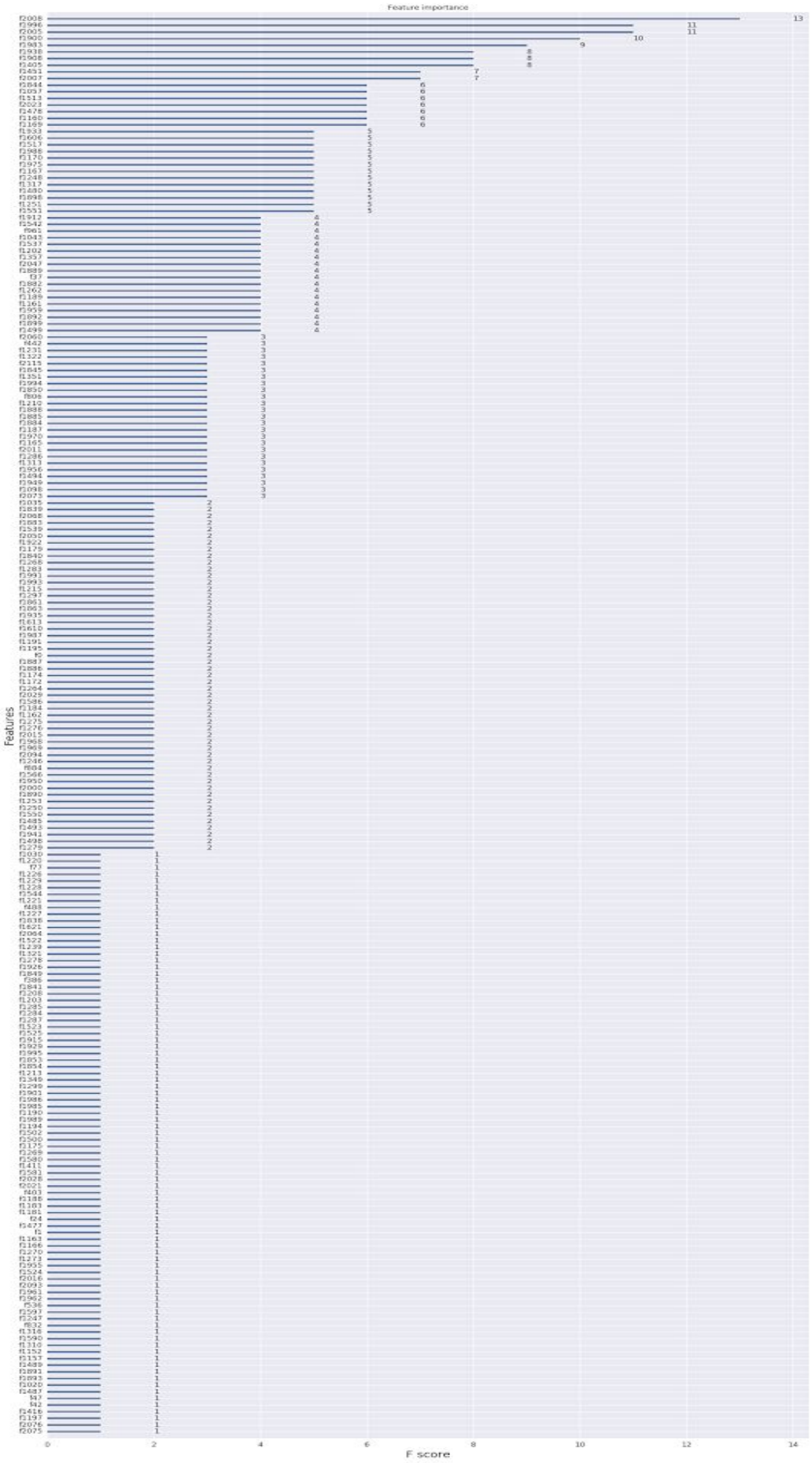

Feature importance
Features
F score
0
2
4
6
8
10
12
14

Fig 5

Fig 5shows the F-score for important features identified by the XGBoost algorithm. Overall the features have relatively low f-scores. The highest value being 13 and the majority of features having f-scores below a value of 3. The shows that a single feature does not strongly correlate with a passing or failing part. Instead, a combination of these low scoring features contribute to predicting whether a part passes or fails.

Reflection

To get to the final XGBoost model I preprocessed the categorical and date data. This was done by encoding the categorical data using the 'leave-one-out' encoding technique and modifying the date data to reflect rolling 24 hour and weekly periods. I initially used the entire data set, but lacked memory to process such a massive data set. To address this issue, I sampled the data set. After sampling and performing an initial model fit to the data I found out that the encoded data and pre-processed weekly date data were not significant features. I ended up not incorporating the categorical data and using the rolling 24 hour date data. I then loaded all the data from the identified important features and tuned the XGBoost parameters to maximize the MCC score.

The Bosch production line data set is 17 GB. It was interesting working with a large dataset that was a mixture of numerical, date, and categorical data. To It provided several interesting challenges that I had not encountered before. I had initially attempted to fit the XGBoost algorithm on my local laptop, but quickly realized it did not have enough processing power. I then used more capable machines on AWS, but even with the added processing power I still needed to find creative solutions to working with the large amount of data. After refining the algorithm I was able to achieve a reasonable MCC score. However, there is lots of improvement to be made before the machine learning system I had developed in predicting a passing/failing part would be used in a production environment.

Improvement

Further improvements that can be made are:

- Determine alternative ways of including categorical data
- Add parallel processing of data to increase speed of fitting XGBoost algorithm
- Use different modeling techniques for categorical, date and numerical data and stacking models

I attempted to incorporate the categorical data and found that it wasn't an important feature in determining if a part would pass or fail in the production line. I think there are other methods that can be used to add the categorical data that can be explored when attempting to improve upon the program I had developed.

Adding parallel processing of the data across multiple machines would improve processing time. It would also allow me to use all the training data, instead of taking a sample of it, thereby improving accuracy of the algorithm.

An additional approach I could use to predicting a passing/failing part would be to use 3 separate learning algorithms for the numerical, date and categorical data. Each of these algorithms could produce its own prediction. The final response would be a weighted calculation of each of the predictions. This method would allow for all the data to be incorporated and would allow for a more tuned approach to looking at each of the data types, instead of using a single monolithic approach to the entire dataset.

Appendix I.

References

1.https://en.wikipedia.org/wiki/Matthews_correlation_coefficient

2. https://en.wikipedia.org/wiki/F1_score

3. https://en.wikipedia.org/wiki/Precision_and_recall

4. http://pubs.acs.org/doi/full/10.1021/ci300348u

5.https://www.kaggle.com/c/bosch-production-line-performance/data

6. http://www.slideshare.net/OwenZhang2/tips-for-data-science-competitions

7.http://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf

8. http://www.slideshare.net/OwenZhang2/tips-for-data-science-competitions

9.http://www.slideshare.net/ShangxuanZhang/kaggle-winning-solution-xgboost-algorithm-let-us-learn-from-its-author

10. http://xgboost.readthedocs.io/en/latest/python/python_api.html

11. https://aws.amazon.com/ec2/instance-types/

12.https://www.kaggle.com/joconnor/bosch-production-line-performance/python-xgboost-starter-0-209-public-mcc