BOSCH PRODUCTION LINE PERFORMANCE

A project report submitted as a part of Machine Learning Course(CS6375)

by

Sasidhar Chennamsetty (Netid : sxc163730)
Sai Vivek Kanaparthy (Netid: sxk163030)
Satya Aravind Chowdary Obellaneni (Netid: sxo160530)
Himanshu Parasahar (Netid: hxp151330)

November 2016

Contents

1 INTRODUCTION **1**
1.1 Overview 1

2 Related Work **2**

3 Dataset Description **3**

4 Pre-Processing Techniques **10**

5 Proposed Solution **11**
5.1 Boosting Algorithm 11
5.2 Advantages of Boosting 11
5.3 XGBoosting 12
5.4 NumPy 12
5.5 Pandas 12

6 Results **14**
6.1 Phase 1: Extraction of Important Features and Labels 14
6.2 Phase 2: Data to the XGB classifier 14
6.3 Phase 3: 3-fold Cross Validation and Accuracy 14
6.4 Phase 4: Testing the Data 14
6.5 Conclusion 16

List of Figures

3.1 Features: 200000 4
3.2 Histogram for the feature L0_S0_F0 5
3.3 Parameter L3_S29_F334 Station L3_S2 variation 5
3.4 Parameter L2_S27_F3206 Station L2_S2 variation 6
3.5 Parameter L0_S9_F160 Station L0_S9 variation 6
3.6 Parameter L0_S7_F138 Station L0_S7 variation 7
3.7 Station Features 7
3.8 Line Features 8
3.9 Stations and Features 9

Chapter 1

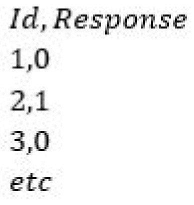
INTRODUCTION

1.1 Overview

Bosch, which is one of the worlds largest and leading manufacturing companies, should always ensure that their production of wide variety manufacturing components should be of highest safety and quality standards. It also monitors and records the data along its assembly lines step by step during the manufacturing process. The problem here is to apply the different analytical techniques on the data obtained during the process of manufacturing and to predict the internal failures during the different steps of the process.
The data of the Bosch Production Line represents the measurements of the parts. Each part is represented by a unique Id. The main goal of the project is to predict which of the parts will fail the quality control i.e., which parts have the Response = 1 in the data set. The submission file accuracy is evaluated on the Mathews Correlation Coefficient (MCC) between the predicted and observed response. The formula for MCC can be given as follows

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

Where TP is the number of True positives. TN is the number of true negatives. FP is the number of False positives. FN is the number of false negatives. For each row in the output file there is part id and the binary prediction response variable corresponding to it. Example output of the submission file:



```
Id, Response
1,0
2,1
3,0
etc
```

Chapter 2

Related Work

In out project, we have used XGBoost[1] technique proposed by Chen et al. In [1] they proposed an efficient tree boosting algorithm for distributed machine learning system. XGBoost[1] can construct parallel trees with higher efficiency. As the dataset is very large consisting of 1157 columns, 80% of the data consists of missing values, some stations have same data values. User beluga has removed the missing values from the data which saves the usage of memory while running the program. And, he has done this process on only on 100000 rows. He also observed that the train and test set has the same period. Lewis used the xgboost on the first 200000 rows of the numeric data to identify important columns leaving the rest of the data as it is more complex to run on a single machine. He found the best numeric features which contribute to the response. Both the users found the best Mathews Coefficient using different threshold values.
Scripus has used the best features from the data and ran the boosting of the few station features. Jeff Delaney found out that the station 32 has the highest error rate and listed the stations with their corresponding features, samples that run through these stations and the error rate. Ronin found the categorical data is lot of sparse where there is no correlation between the numeric and date data.
In our project, we made use of the pandas module of python to read the large data. We determined the correlation between the different stations. Since the data is huge we ran the 3-fold cross validation of the data and determined the accuracy and ROC. We tried finding different thresholds and used the best Mathews Coefficient to perform analysis on the test data.

Chapter 3

Dataset Description

The dataset contains vast number of anonymized features. The naming convention is as follows For example: L2_S32_F3838 is measured on line 2, station 32 and is feature number 3838. The train dataset contains three categories of files:

- Test_categorical.csv
- Test_data.csv
- Test_numeric.csv

The test dataset contains three categories of files:

- Test_categorical.csv
- Test_data.csv
- Test_numeric.csv

The data feature gives the timestamp about the time when each measurement was taken. Every date column ends in a number that corresponds to the previous feature number.

For example: The value of L0_S0_D1 is the time at which L0_S0_F0 was taken. Train_numeric.csv and Test_numeric.csv contains the response variable where the response is stored.

More than 80% of the data contains missing values (na values) which made the data very sparse and the size of the data is very huge to run the classifier. As the data is sparse the correlation plot on the whole data can be observed as follows:
The correlation plot taking 20000 instances is as follows: Not much can be inferred from the above plot but we can see that the classifier should run on the whole data rather than subset of the data.

Now for the Visualization of parameters over time. The following plots shows the visualization of some the station parameters with the features over time. These plots show variance of the numeric measurements of parts produced at that tool. From the plots we can see that failing parts are marked red (*) .

Table 3.1: Dataset Description

Data Set	Number of Features	Number of Instances
Train_numeric.csv	970	1183748
Test_numeric.csv	970	1183748
Train_categorical.csv	2141	1183748
Test_categorical.csv	2141	1183748
Train_date.csv	1157	1183748
Test_date.csv	1157	1183748

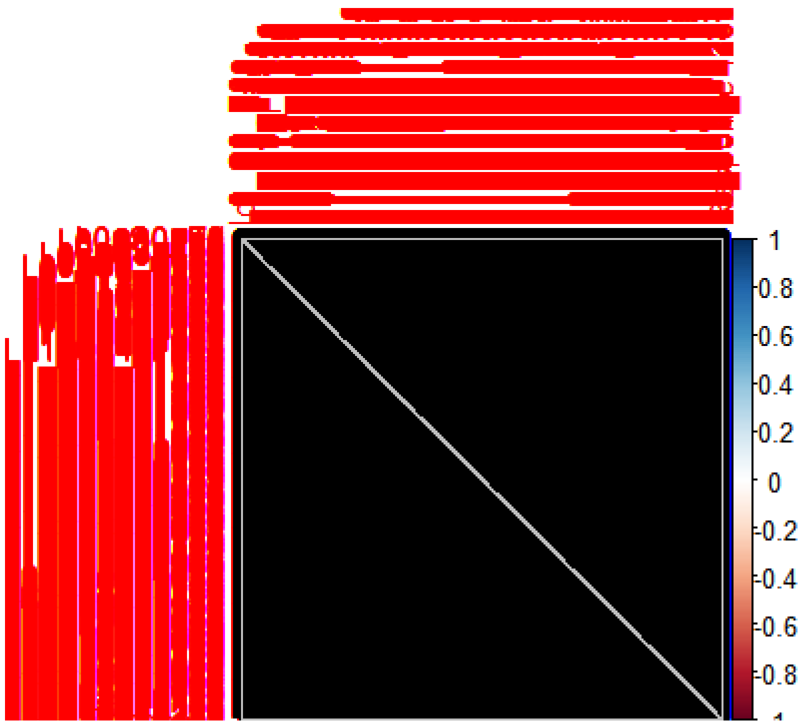


Figure 3.1: Features: 200000

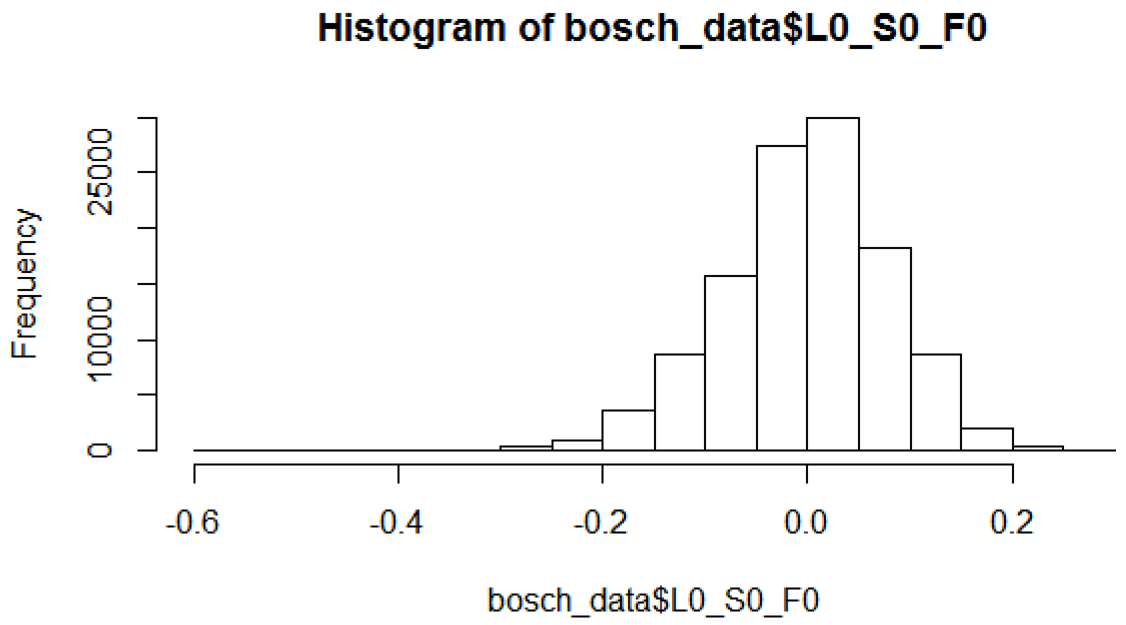


Figure 3.2: Histogram for the feature L0_S0_F0

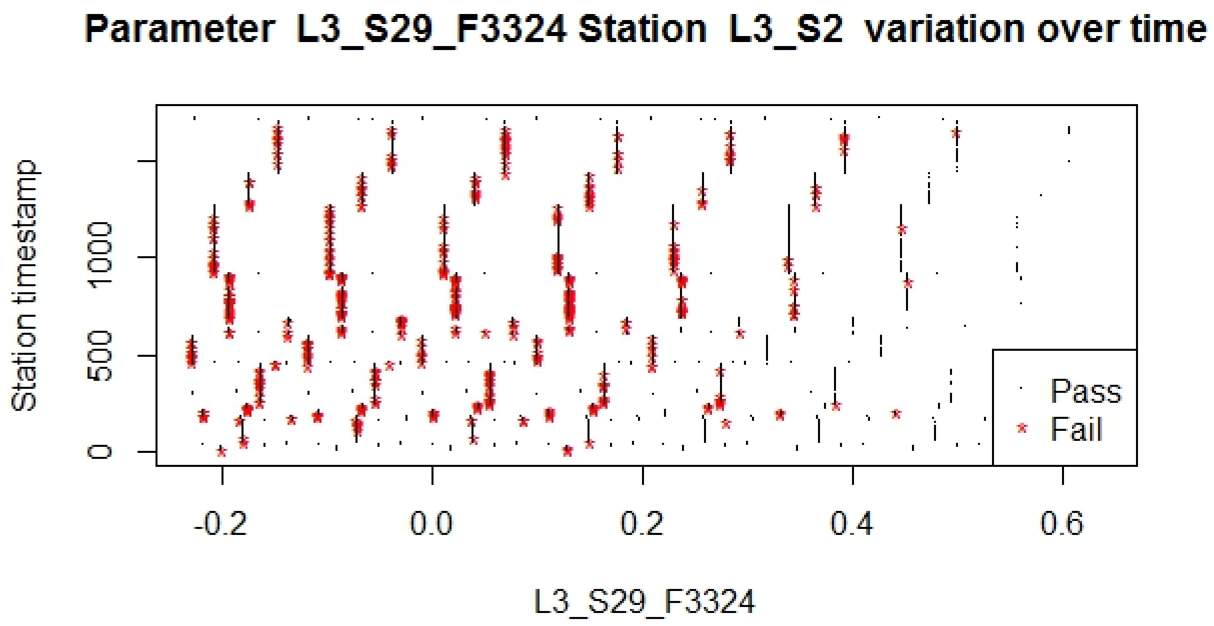


Figure 3.3: Parameter L3_S29_F334 Station L3_S2 variation

To visualize the data, the feature dependency function in the code determines the line features and station features.
The screen shot of the output of station features is as follows

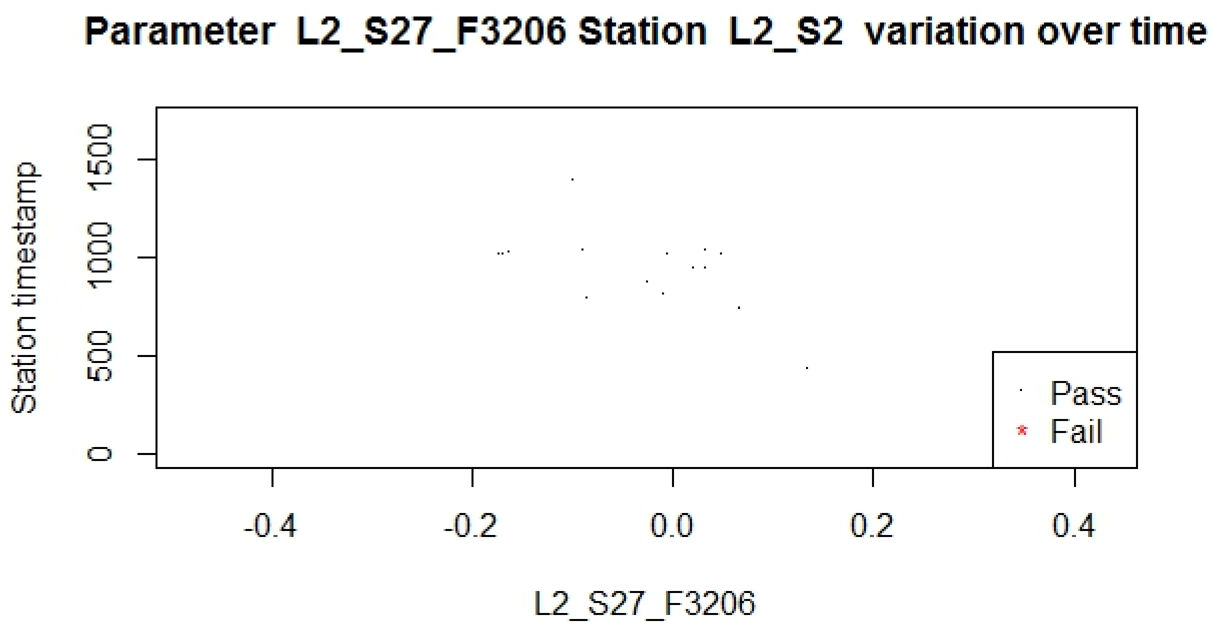


Figure 3.4: Parameter L2_S27_F3206 Station L2_S2 variation

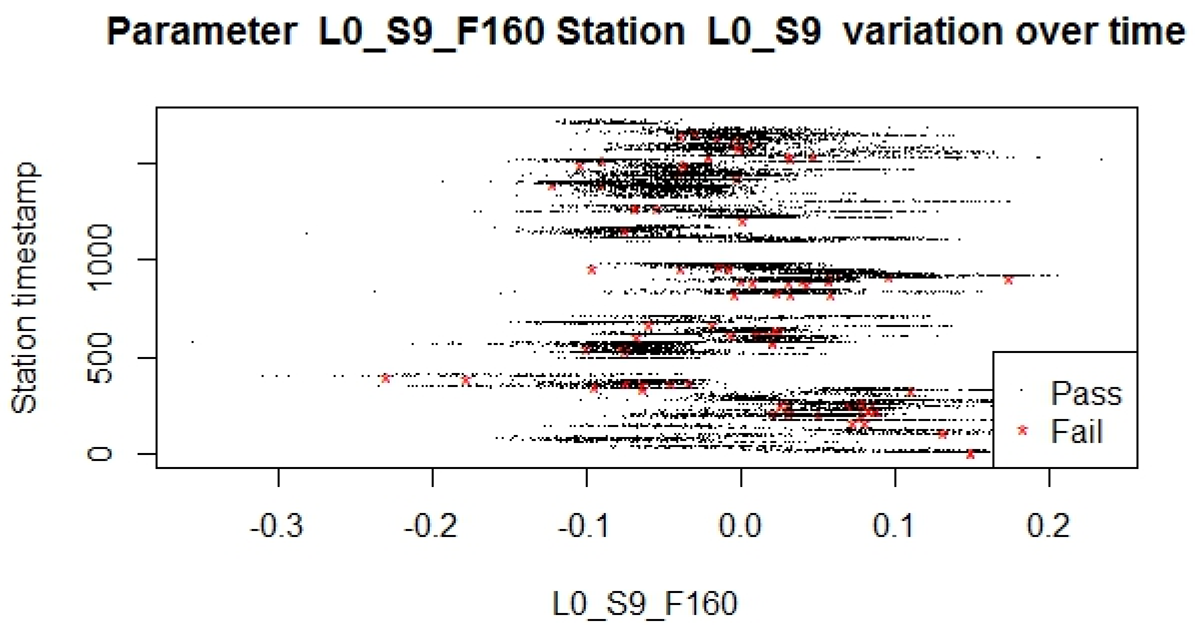


Figure 3.5: Parameter L0_S9_F160 Station L0_S9 variation

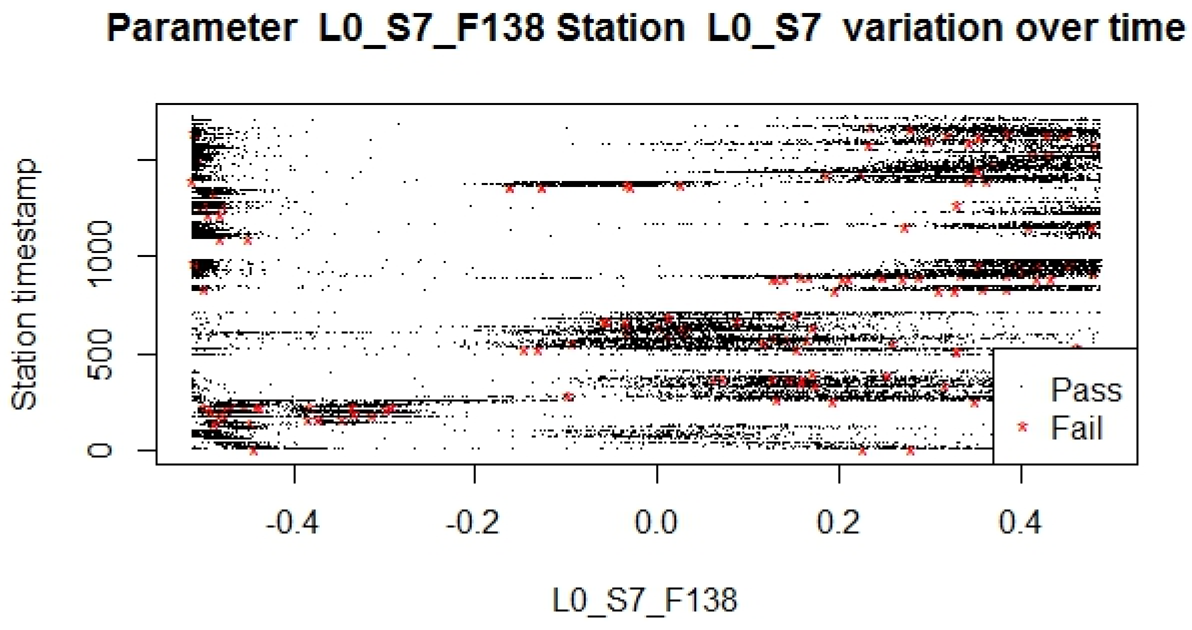
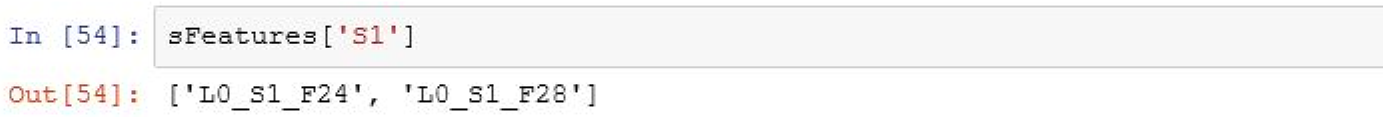


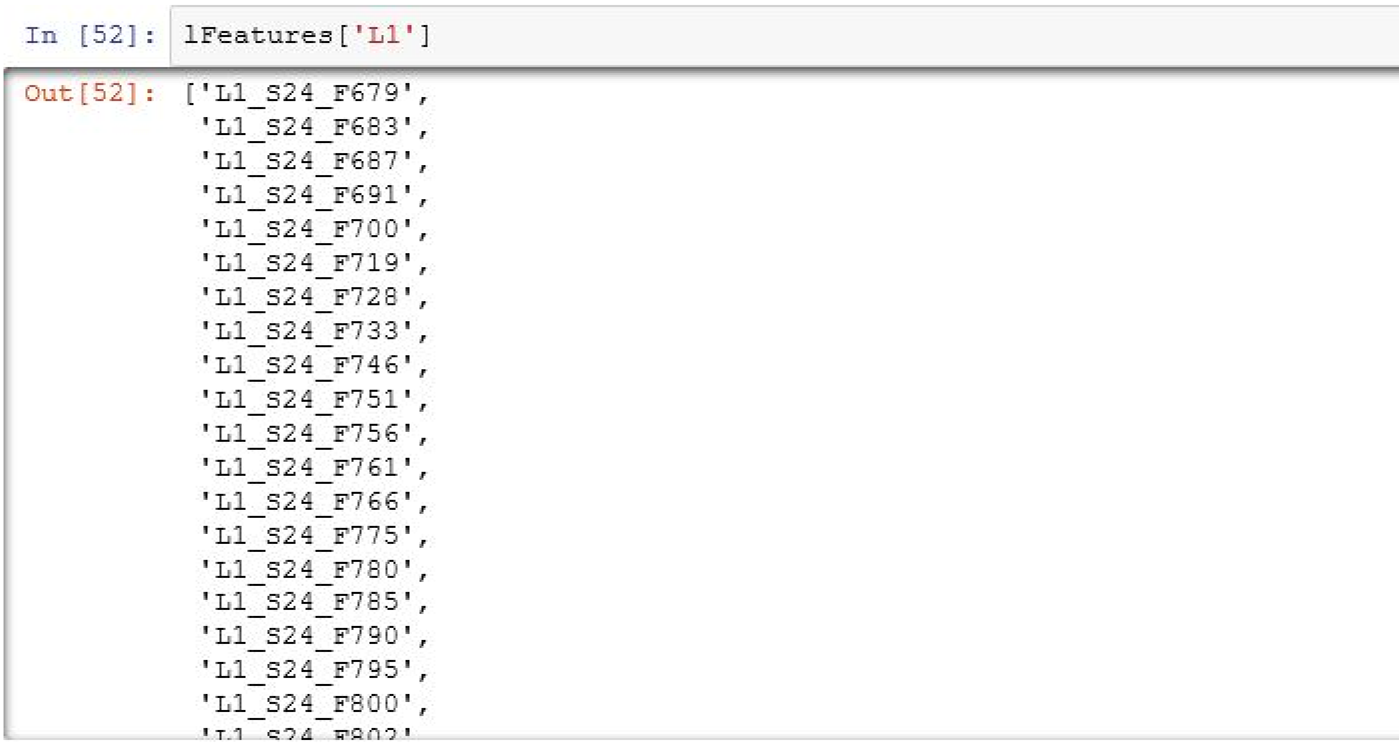
Figure 3.6: Parameter L0_S7_F138 Station L0_S7 variation



```
In [54]: sFeatures['S1']
Out[54]: ['L0_S1_F24', 'L0_S1_F28']
```

Figure 3.7: Station Features

The screen shot of the output of line features is as follows:

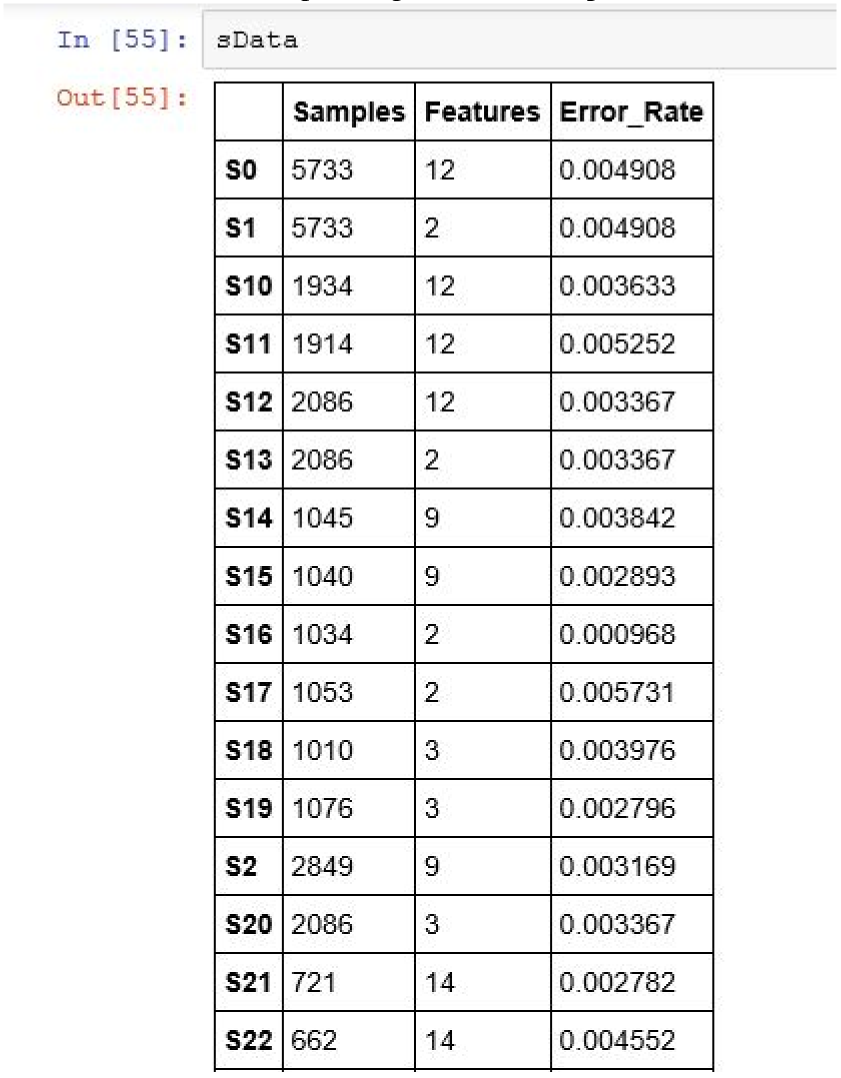

```
In [52]: lFeatures['L1']
```

```
Out[52]: ['L1_S24_F679',
          'L1_S24_F683',
          'L1_S24_F687',
          'L1_S24_F691',
          'L1_S24_F700',
          'L1_S24_F719',
          'L1_S24_F728',
          'L1_S24_F733',
          'L1_S24_F746',
          'L1_S24_F751',
          'L1_S24_F756',
          'L1_S24_F761',
          'L1_S24_F766',
          'L1_S24_F775',
          'L1_S24_F780',
          'L1_S24_F785',
          'L1_S24_F790',
          'L1_S24_F795',
          'L1_S24_F800',
```

Figure 3.8: Line Features

It can be deduced that the Station 32 has a dependency only on one feature. i.e., L3_S32_F3850.

The station data with the corresponding features, samples and error rate can be visualized as follows:



```
In [55]: sData
Out[55]:
```

	Samples	Features	Error_Rate
S0	5733	12	0.004908
S1	5733	2	0.004908
S10	1934	12	0.003633
S11	1914	12	0.005252
S12	2086	12	0.003367
S13	2086	2	0.003367
S14	1045	9	0.003842
S15	1040	9	0.002893
S16	1034	2	0.000968
S17	1053	2	0.005731
S18	1010	3	0.003976
S19	1076	3	0.002796
S2	2849	9	0.003169
S20	2086	3	0.003367
S21	721	14	0.002782
S22	662	14	0.004552

Figure 3.9: Stations and Features

Chapter 4

Pre-Processing Techniques

Since the data is huge, to load the large dataset we used the pandas module in python using the chunk size parameter as 100000. As the train_categorical.csv is a sparse data with lot of missing values, it does not contribute to the output. So, the classifier depends on the train_date.csv and train_numeric.csv. We extracted the labels from the train_numeric data. Now after loading the data we ran the loop to take the feature values from the train_numeric data and train_date data. Now using these we trained the data using the XGB classifier module in python and we took the values which are crossing a threshold value. Due to the above step the complexity of the data is greatly reduced.
To get rid of the nan values and the data is very sparse instead of directly cleaning the data we gave the data to the classifier and evaluated the importance of each feature to a certain threshold value select and features having feature importance greater than the threshold. Those features are being considered for the further evaluation of the failure rate.
In the pre-processing phase, the main aim was to get rid of NaN values. Because of these NaN values, the matrix turns out to be very sparse. The data is not directly cleaned to remove NaN values instead the complete data is given to the XGBClassifier to create a model. This model gives the feature importance of all the features. Threshold is being set to differentiate between the features which are not useful i.e., features having majority of its values as NaN and the features which are important. The threshold is set to 0.0045.

Chapter 5

Proposed Solution

5.1 Boosting Algorithm

Boosting is a type of algorithm which convert weak learners to strong learners in Machine learning. It combines the prediction of each weak learner using methods like

- Average/Weighted Average
- Considering prediction has higher vote

So Boosting combines the weak learners to form a strong rule. And weak rule/learner are identified by applying base learning algorithms with different distribution which generates a new weak prediction rule each time. Each time a different distribution is chosen by

- All distribution is taken and equal weight is assigned to each.
- We pay higher attention to observations If first learning algorithm causes prediction error and then next learning algorithm is applied.
- Step2 is repeated till limit is not reached for base learning algorithm.

With these iterations boosting combines their weak rules into a single strong prediction rule. There are multiple types of boosting algorithms like AdaBoost i.e Adaptive Boosting, Gradient Boosting or Gradient Tree Boosting and XGBoost. In our project we have implemented XGBoost. XGBoosting is efficient and most reliable package that can be used for implementing Gradient Boosting algorithm over large datasets. It serves as an alternative to run gradient boosting on highly distributed environments like Hadoop.

5.2 Advantages of Boosting

- Most of the Boosting algorithms iteratively learn weak classifiers and add them to final strong classifier and they are weighted in a way that it usually relates with that weak learners accuracy so the weak learners which will come future in iteration focus more on the examples that previous weak learners misclassified

- It has Good generalization and doesnt overfit
- Computation time is more than SVM or other classifier but Accuracy is more
- Can handle large data and can predict better.

5.3 XGBoosting

XGBoosting is predominantly used module for boosting in python. It implements extreme gradient boosting. It is also known as regularized boosting technique. It handles missing values by its inbuilt routine. It is more easy to get optimized results by using XGboost because it allows the user to run cross validation at each iteration. Internally XGBoost builds a gradient boosted trees by itself in a parallel fashion which helps to gain more performance than any other ensemble methods. It can load the data from numpy package. The data is stored in the DMatrix object. The parameters can be set to classifier while classifying the data. The parameters used for the data are objective which is set to binary:logistic and maxdepth is set to 3. The train method in the xgb module used to train the data.

5.4 NumPy

NumPy is one of the most important and fundamental package for computing various complex algorithms using Python. It has a key feature of n-dimensional array which stores data as tables. Before using NumPy in your project, we need to import this package. In our project, we used NumPy in method sigFeatures where it is used to find values based on our condition given Ex: numpy.where($x > 0.5)[0]$. We created multi-dimensional array of float type ”dtype” by using numpy.float32. There are numerous methods in this package, we used numpy.concatenate, numpy.float, numpy.where etc. Some advantages of NumPy are, It has Array objects (scalars, dtypes, ndarrays), Universal functions(Broadcasting, error handling, casting rules, using internal buffers), Routines(Array manipulation routines, binary string operations) etc.
NumPy’s arrays are more efficient than python lists because of acquiring less memory and time. Accessing data from NumPy’s arrays are much faster than python lists. While working on statistical analysis, histograms and visualization libraries NumPy’s arrays serves the best.

5.5 Pandas

Pandas is a python module which provides fast and efficient data structures designed to work with labelled and relational data. It is one of the key module in python used for real world data analysis. Pandas is built on numpy which is used to integrate well with scientific computing. As the data is huge and to process such huge data we made use of pandas module of python. Pandas python module provides efficient access of huge data.
The csv module in python does not provide efficient access to the huge data like the pandas module. We used two methods pandas.read_csv and pandas.io.parsers.read_csv to read the data. pandas.read_csv method is faster than the pandas.io.parsers.read_csv. Chunk size can be choosen using the parameter chunksize in read_csv. The read_csv method is ran over the

chunk_size of 100000. pandas.concat method is used to concat along the particular axis. If the axis is set to 1 it is concatenated along the rows and if the axis is set to 0 it is concatenated along the columns.

- Pandas is one of the most efficient for reading/writing csv files and manipulating the data with integrated indexing.
- It has the tools for reading and writing the data both structured and unstructured.
- It has high performance for appending and merging of data sets.
- Axis indexing provides an efficient way for working with higher dimensionality in a lower dimensionality data structure

Chapter 6

Results

6.1 Phase 1: Extraction of Important Features and Labels

In Phase 1 we extracted the main features from the data. As the data is sparse and huge it takes more time just to read the data. So, to make it more fast we extracted the important features where the samples and features are highly dependent on the stations. Initially we combined the categorical data and date data. Now we generated the important feature indexes. After finding the dependency when we ran the analysis the entire data with the important features the complexity is greatly reduced since 80% of the data is sparse.

6.2 Phase 2: Data to the XGB classifier

Now from the important features generated from the phase 1 we train the classifier by tuning different parameters and select the best parameters to give best complexity time. Then the classifier is fitted with the feature values and labels. Now we generated a threshold value form the Mathew Correlation Coefficient function for better classification.

6.3 Phase 3: 3-fold Cross Validation and Accuracy

After the classifier generated the features we performed 3-Fold Cross Validation and determined the accuracy using ROC metric. The ROC values are as follows

6.4 Phase 4: Testing the Data

Now from the generated features the testing data is passed to XGB classifier to generate the predicted values. The predicted values are stored with responses in a csv file. The predicted values are as follows:

- Number of data entries for Response=0 are 1179444

Table 6.1: ROC Metric

Iteration	Parameter: max_depth	Parameter: base_score	Parameter: colsample_bytree	Average ROC
1	3	0.005	0.5	0.631
2	3	0.05	0.4	0.647
3	5	0.005	0.5	0.651
4	5	0.05	0.4	0.656
5	7	0.005	0.5	0.671
6	7	0.05	0.4	0.664

Table 6.2: ROC Metric

Fold	ROC Values
1	0.664
2	0.647
3	0.645

Table 6.3: Accuracy Metric

Iteration	Accuracy
1	73.1%
2	70.4%
3	71.3%

- Number of data entries for Response=1 are 4304

The failure rate can be inferred from the above responses as follows:
The failure rate = (4304/1183748)*100 = 0.36%

6.5 Conclusion

The agenda of the Bosch production line performance contest is to monitor its path as they progress through the manufacturing process. In this project, we have analyzed the relationship between the features across the stations and lines. A very interesting result, that the station 32 was found to depend only on one feature i.e., L3_S32_F3850.
As the size of the data is very large, pre-processing is done. The feature dependency is evaluated across stations and lines. A threshold value is set to determine the importance of the features. The features which score above the threshold as an output of the classifier are considered important. The XGBoosting Algorithm is then being applied on the important features varied across different parameters and the best parameters are being evaluated. A k-fold cross validation is taken as a measure to evaluate the performace of various parameters along with the accuracy.

- The average ROC obtained is 0.652
- The average training accuracy obtained is 71.59%
- The failure rate obtained is 0.36%

Bibliography

[1] Chen, Tianqi, and Carlos Guestrin. ”XGBoost: Reliable Large-scale Tree Boosting System.”.

[2] http://pandas.pydata.org/pandas-docs/stable/

[3] https://pypi.python.org/pypi/xgboost/

[4] https://docs.scipy.org/doc/numpy/

[5] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html

[6] http://scikitlearn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

[7] http://matplotlib.org/api/pyplot_api.html

[8] http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

[9] http://scikit-learn.org/stable/modules/cross_validation.htmlstratified-k-fold

[10] http://seaborn.pydata.org/tutorial.html