BOSCH PRODUCTION LINE PERFORMANCE MEASURE

PROJECT REPORT

Team Members:

Shobhika Panda
Srinivas Lingamgunta
Mahima Jayaprakash
Lahari Ganesha
Jayaprakash Rout

OBJECTIVE:

To predict internal failures using thousands of measurements recorded by **BOSCH** at each processing step 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.

DATASET DETAILS:

Train_Numeric:
Number of instances = 6471
Number of features = 970 (Including ID + class variable)

Test_Numeric:
Number of instances = 1183749
Number of features= 969(absence of class variable)

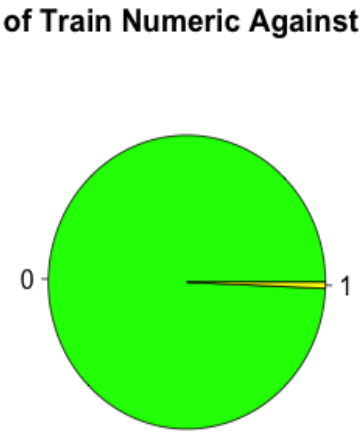
Train_Date:
Number of instances = 1183748
Number of features= 1157

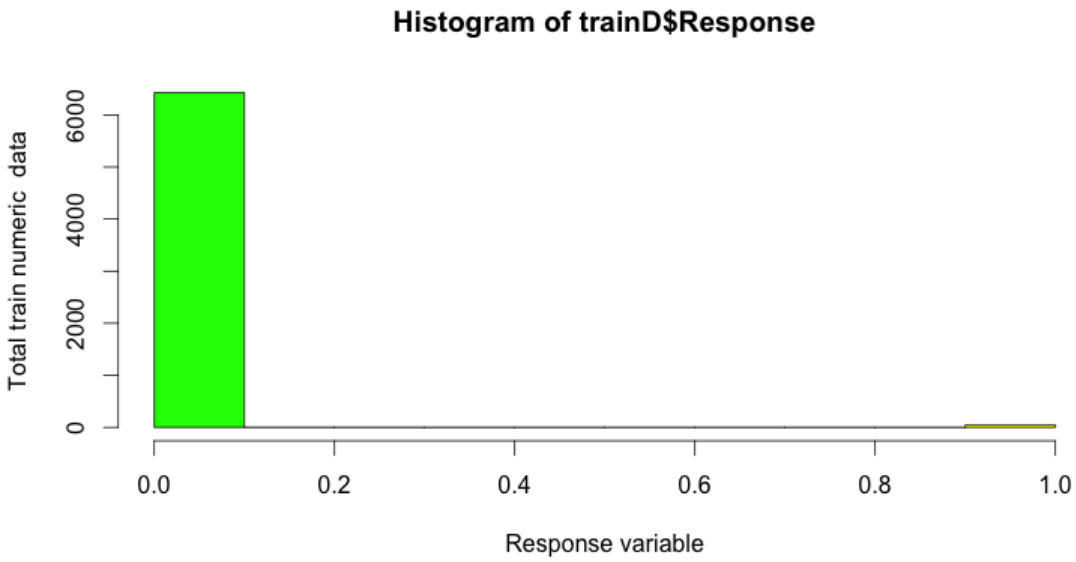
Test_Date:
Number of instances = 1183749
Number of features= 2141

DATA SOURCE

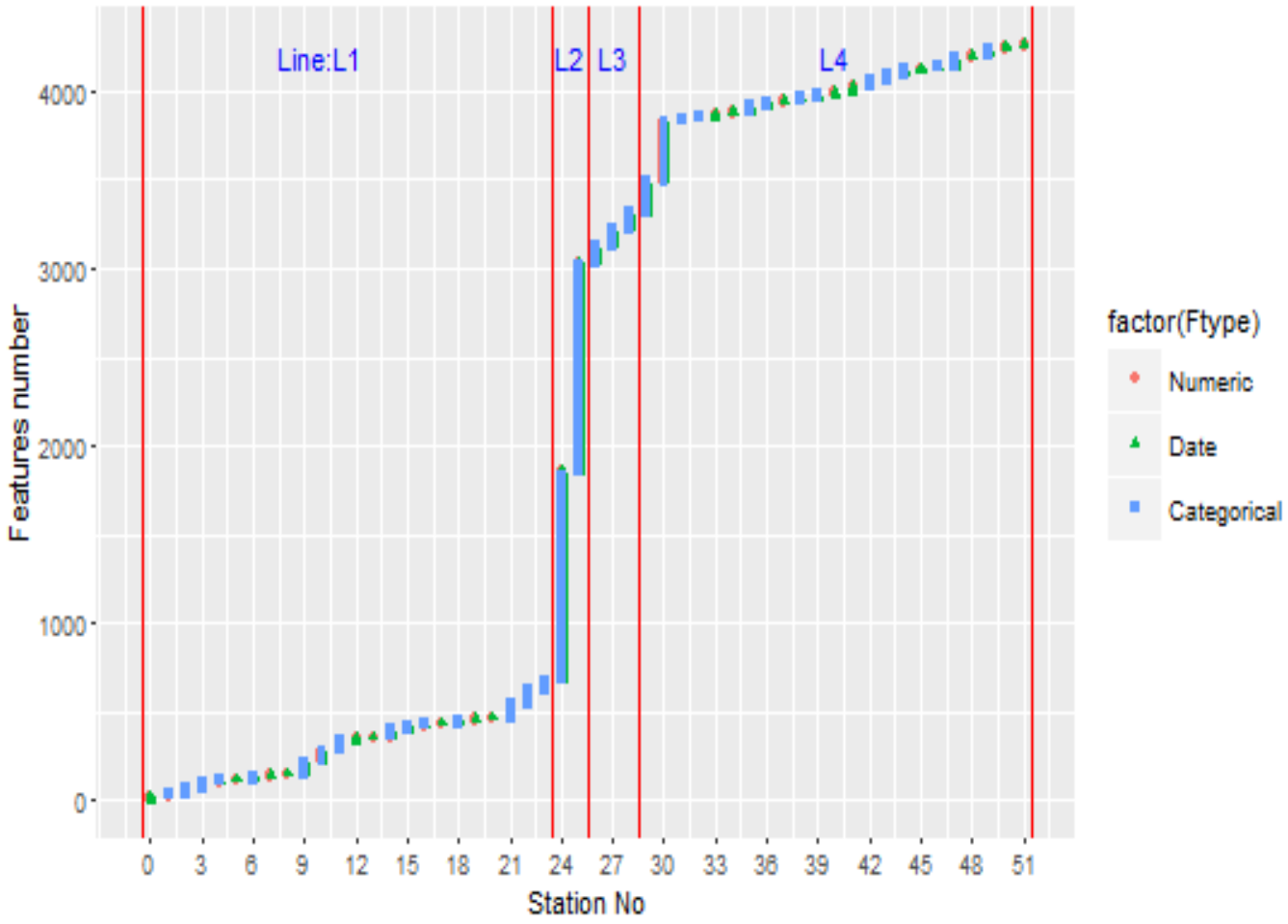
https://www.kaggle.com/c/bosch-production-line-performance

INITIAL DATA DISTRIBUTION





ggPlot of the Features against the Station numbers:

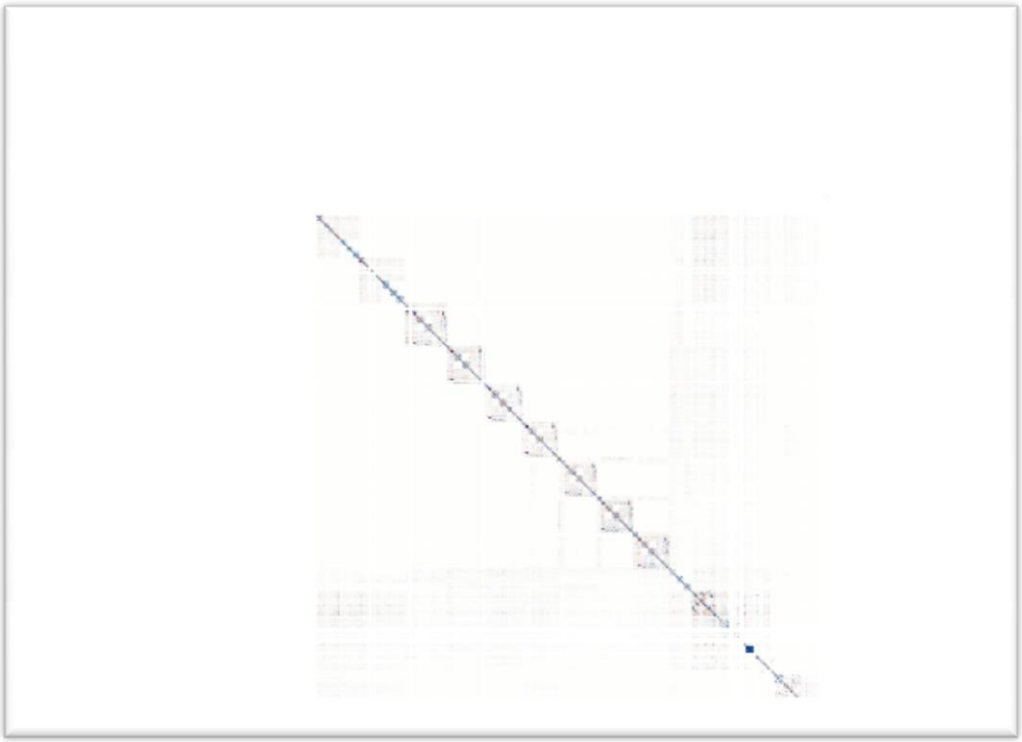


By plotting, we analyzed how various features are distributed. Clearly Stations 24 and 25 are having the highest features 1177 and 1184. This can be found below:

Observations: 4,264 Variables :5

```
## $ features <fctr> L0_S0_F0, L0_S0_D1, L0_S0_F2, L0_S0_D3, L0_S0_F4, L0...
## $ Ftype    <fctr> Numeric, Date, Numeric, Date, Numeric, Date, Numeric...
## $ Line     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ Station  <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ Fno      <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,...
```

CORRELATION:



The above plot captures the correlation between the features and the class variable of train_numeric dataset.

TECHNIQUES USED:

We conducted the experiment using multiple classifiers:

- **Decision Tree Classifier**
- **Random Forest Classifier**
- **Extreme Gradient Boosting Classifier**

PREPROCESSING:

i) We have data which is largely comprising of null values. Firstly, we made an offset of 2 to the values. This will differentiate the original numerical values from null values.

ii) Secondly, we made all the null values to zero so that would help in eradicating errors while building the model.

Prevent overfitting:

- Since we are performing the extraction of the important features, we are ignoring the less important features for more accuracy and efficiency. Around 36 features have been considered after pre-processing using XGB.

ALGORITHM UNDERSTANDING:

1) **Decision Tree Classifier**: It performs greedily and recursively by binary partitioning of features.
   It predicts label for leaf. Each partition is selected greedily. We select the best split from the set of possible splits in order to maximize the Information Gain at the node. We chose the split by maximum Information gain. We find the node impurity by using Gini Impurity which is a measure of homogeneity of the labels and the node.

2) **Random Forest Classifier:** It is the ensemble method of decision trees. It is considered to be more successful for classification. It combines several decision trees in order to reduce the overfitting. Random Forest train a set of decision trees separately so that the training is done in parallel. It injects randomness and aggregates the results of all the random trees. For classification, the prediction is based on the majority vote. Each tree prediction is considered as the vote for one class. The training is based on sub-sampling to get a different training set every time. It also considers different random subsets of features to split on at each node.

3) **Extreme Gradient Descent classifier:**.

   We divided the training dataset into two models, first part for modelling (dmodel) and second part for validation(dvalid). eXtreme Gradient Boosting Training (xgb) is used in training. Here we use the feature of watchlist wherein we specify for each round what parameter the model should take into consideration before performing the boosting for round 2. We use ROC as the evaluation metric for each round of evaluation performed while building the model.

   Then we use the importance matrix building feature of the xgb (xgb.feature), By constructing the importance matrix, we get the important features which helped in building the model. The important features derived from here are determined with the help of the gain attribute of each feature in the model. Matthews correlation coefficient (MCC) is a measure which takes into account true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes.

   We then performed 10-fold cross validation on the data set by changing the model and validation parts on the training set. We finally did an average of the MCC obtained on each validation. Final MCC for this set is 0.995. Using the obtained MCC, we are able to select a best threshold to label cases as positive, this helps in building an optimum threshold instead of using the direct probability or a specific cutoff. We are setting the predicted variable to positive only after it exceeds the best threshold which in our case is 0.995.

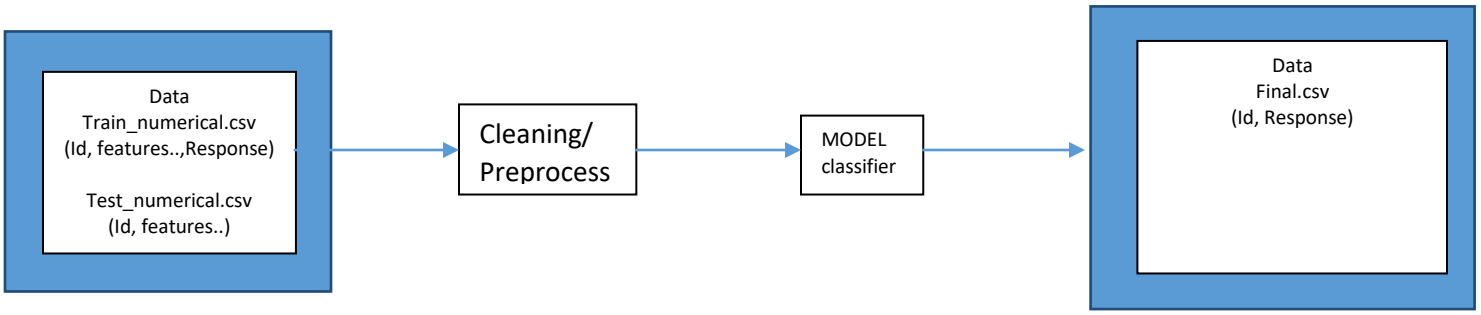
   Further we computed the accuracy of the model and also computed the precision and recall. Our accuracy was 99.21 for 200,000 test records. Our data was hugely biased towards negative than positive. This prompted us to also use other performance evaluation

metrics. We also computed recall and precision to better validate our model. The precision we got was 79.72 and the recall was 55.63 which proves this model has good performance metrics.

EXPERIMENTAL METHODOLOGY:

- First we have the raw dataset downloaded from the above mentioned resource.

- We preprocess the data in order to reduce the size and get rid of unnecessary data:
  - Convert NULL(NaN) values to 0 value of the column
  - Considering the important features by implementing xgb classifier with the response variable.
- On these obtained important attributes use XGB model, Decision Tree and Random Forest classifiers to learn the train data and predict the response values for test data.
- Since the data is unevenly classified, precision and recall will help us determine how strong is the model.



ACCURACY FINDINGS:

Classifier	Accuracy	Precision	Recall
XGB	99.1	78.3	51.85
DecisionTree	98.5	63.66	48.66
RandomForest	99.3	66.66	40.55

EXPERIMENTAL EVALUATION:

Matthews Correlation Coefficient: Used this evaluation metric as a measure of quality of binary classification. It takes into account true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes. The MCC is in essence a correlation coefficient between the observed and predicted binary classifications; it 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.

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

With a fixed threshold, it finds the best score.

-We are also using Test error to test the accuracy in Random Forest Classifier.

Accuracy = 99.3%

TestError = 1 – Accuracy.

For XGB Classifier, we have got the precision = **78.3%**

Coding Language used:

1. Apache Spark - Scala
2. R for Data analysis and visualization and pre processing.
3. ML lib Classification

FUTURE ENHANCEMENTS:

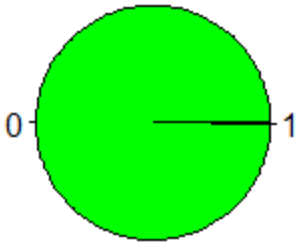
We can work on enhancing the efficiency and accuracy by pre- processing data at an extreme level by checking outliers and doing PCA dimension reduction. This can be extended by implementing more classifiers like Bagging in order to get better insights of the performance measure.

CONCLUSION:

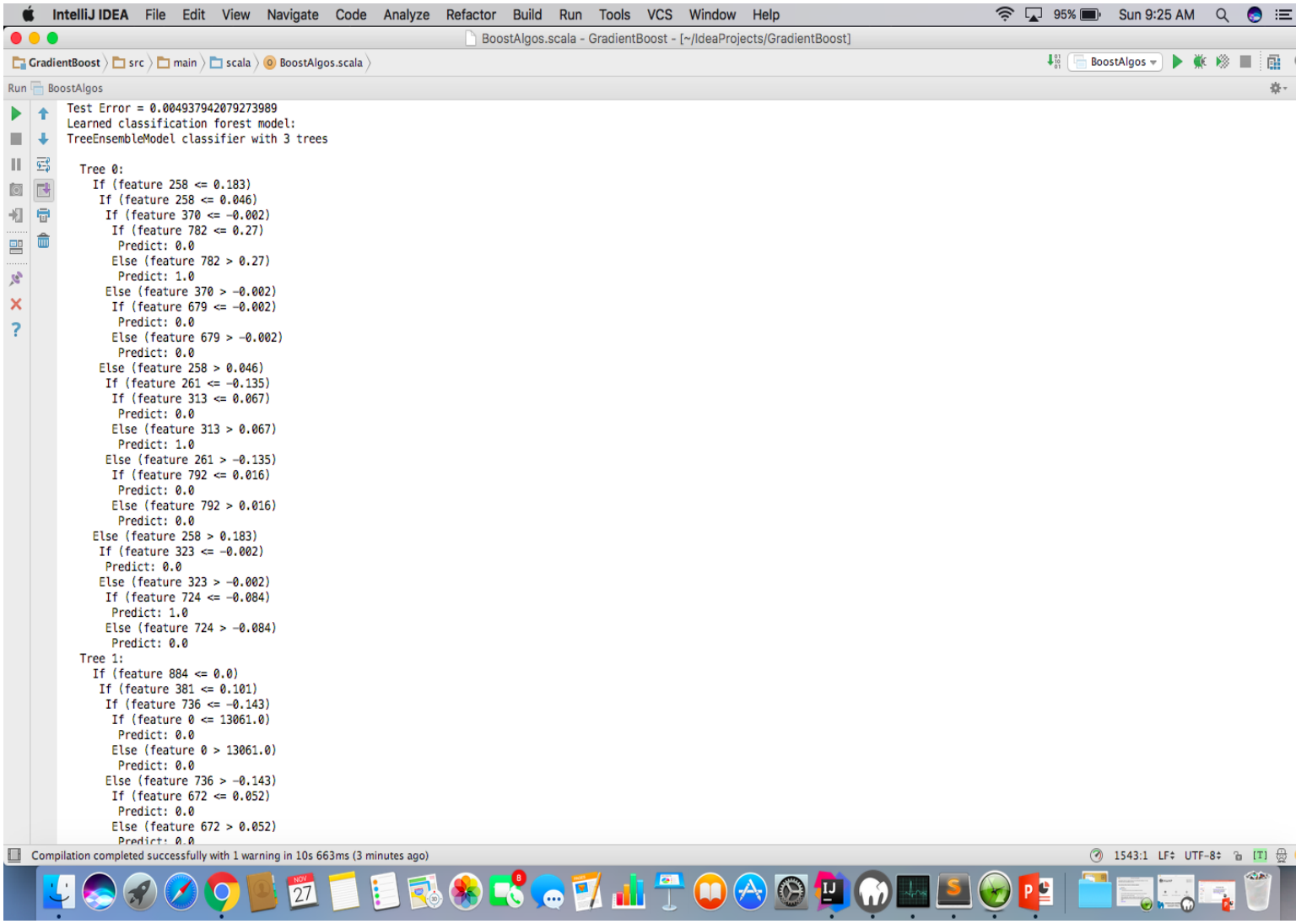
By analyzing different models on the dataset, we identified that accuracy wasn't the best parameter to measure because the data is not equally distributed and heavily biased towards false class. We analyzed the models based on the precision and Recall. By observing the precision and recall, we see that XGBoost model performs better because in Random Forest we had to limit the tree construction to a reasonable depth due to time complexity and the attributes considered for the tree weren't sufficient to produce a better result as this data was dependent on many attributes. On the other hand, XGBoost computes model by boosting with iterations each time.

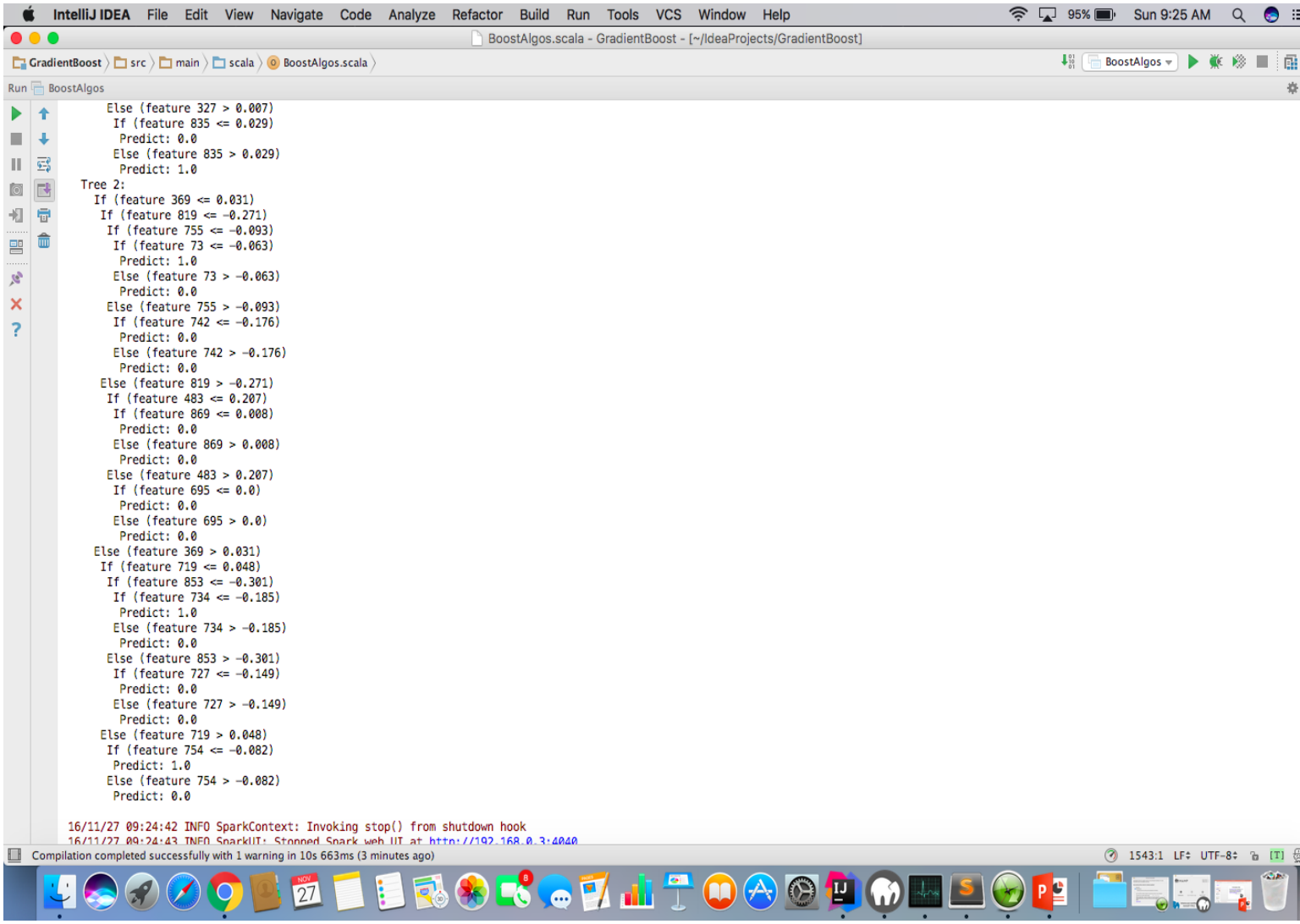
RESULT:

Pie Chart of Test Numeric Response



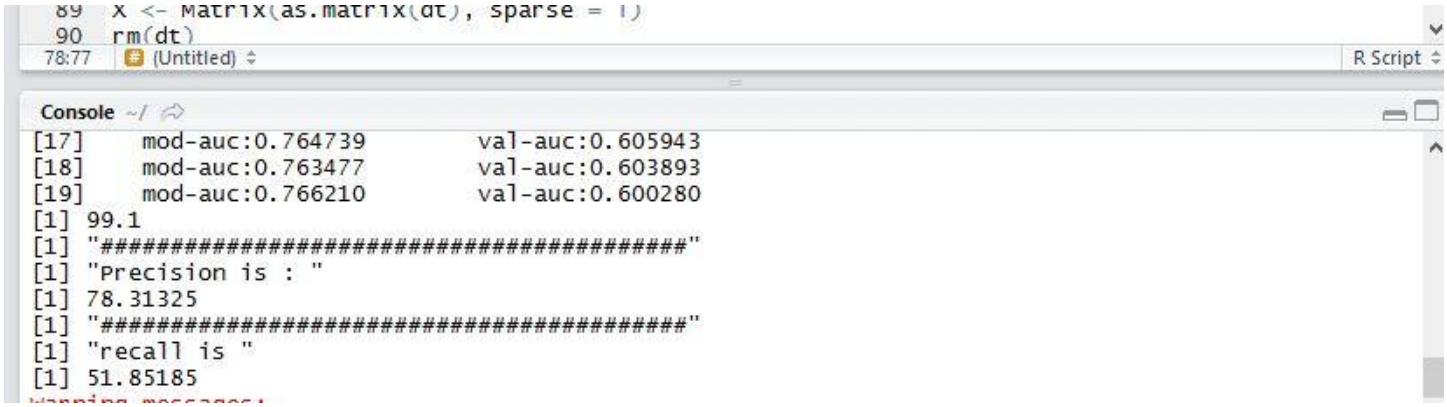
Result of Random Forest Classifier:
Test Error = 0.007477711688832453
Accuracy of Test = 99.3 %
TreeEnsembleModel classifier with 3 trees





Output of Decision tree: 98.5 % accuracy.

Output for XGB:



We have generated the final results in the form of a csv file which contains Id and the corresponding response as shown below:

Id	Response
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	1
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0

BIBLIOGRAPHY:

- ➔ Spark MLib Documentation- https://spark.apache.org/docs/latest/ml-guide.html
- ➔ XGB Classifier - http://xgboost.readthedocs.io/en/latest/R-package/xgboostPresentation.html
- ➔ MCC Correlation evaluation
  - https://www.kaggle.com/c/bosch-production-line-performance/details/evaluation
  - https://www.kaggle.com/cpmpml/bosch-production-line-performance/optimizing-probabilities-for-best-mcc/comments
- ➔ Decision Tree Classifier - http://spark.apache.org/docs/latest/mllib-decision-tree.html