PROJECT REPORT

AUTO INSURANCE CLAIM PREDICTION

TEAM NAME: SUBRAMANIYAN_VENKATARAMANAN_NARASIMHAN_GURRAM

TEAM MEMBERS

ARCHANAVENKATARAMANAN (M12511297)

MRIDULA NARASIMHAN (M12511190)

JANANI SUBRAMANIYAN(M12484583)

GURRAM VENKATA SUBRAHMANYA ADITYA(M12485180)

1.INTRODUCTION

Predictive modelling is a mathematical/computational method that is used in machine learning to predict an outcome. Insurance prediction is one of the widely explored problem in the field of Machine Learning. In that context, insurance industries are investing more in such predictive modelling approaches to increase their profit margin. The goal of this project is to develop a model to predict auto insurance claim for the next annual cycle.

1.1 MOTIVATION

Imprecision in predicting the insurance claim for the successive year increases the insurance cost for safe drivers and decreases the cost for unsafe drivers. The challenges in predicting a potential claimer and the complex dataset involved in this process was the main motivation behind taking up this topic. The interesting part of the project is handling large dataset with anonymous features, missing values and class imbalance.

2. BASIC APPROACH

The model developed in this project is a part of ongoing Kaggle competition. This model is used to predict the probability that a driver will initiate an auto insurance claim. The dataset for this model is provided by Porto Seguro, an auto and home insurance company.

The data consists of,

1. Training data values → Various features for building the model
2. Training data labels → The class label for each training data
3. Testing data values→ the variables that require predictions

For the proposed approach, the following machine learning classifiers were implemented.

2.1 RANDOM FOREST CLASSIFIER

Random forest classifier is an ensemble learning method suitable for classification and regression [2]. This classifier uses diverse decision trees by introducing randomness in the classifier construction and predict the classification by taking the mode of all the output classes. Splitting of node while constructing the tree is by considering the best split among the random subset of the features. The randomness in the classifier causes high bias, but the averaging of results causes the variance to decrease. This bias-variance trade-off yields a better model.

Parameters

1. oob_score → A flag to choose out of bag sample features for better results.
2. random_state → A value used by random number generator.
3. No_of_trees → Large number of trees are better.
4. n_jobs → Number of jobs to run in parallel.
5. min_samples_leaf → Least number of sample features at the leaf node.

2.2 LIGHT GRADIENT BOOST

Light gradient boosting method is a distributed tree based algorithm. Unlike other tree based algorithm, this classifier constructs tree vertically. Thus, it is faster and consumes only very limited amount of memory. It can also handle large amount of data and it mainly focuses on accuracy.

Parameters

1. max_depth →Calculates maximum depth of the tree and handles overfitting.
2. learning_rate → Shrinkage rate determines the impact of each tree on the outcome.
3. feature_fraction → Randomly selects a percentage of parameters for building trees in each iteration.
4. bagging_fraction → Speeds up the training and avoids over fitting.
5. min_split_gain →It controls the number of useful splits in the tree.

2.3 XTREME GRADIENT BOOST

Xtreme Gradient Boost algorithm is a supervised machine learning algorithm based on tree ensemble model. It has built in cross validation feature and thus produces optimum number of boosting iterations [8]. The prediction accuracies are the summed-up value of each individual trees.

Parameters

1. max_depth → Calculates the maximum depth of the tree and handles overfitting
2. silent →True/ False, Boolean value which sets the running messages to be printed
3. objective→ binary: logistic – Logistic regression for binary classification. It returns the predicted probability
4. eval_metric → Area under the curve used for validation
5. learning_rate→ Shrinks the contribution of each tree 0.1 is default

2.4 SUPPORT VECTOR MACHINE

It is a supervised machine learning algorithm used for both classification and regression. Classification is performed by plotting the hyperplane that differentiates classes. Kernels are used to map data from low dimensional space to high dimensional space [3]. This helps in converting a non-separable problem to a separable problem. The following kernels are used in the proposed approach.

1. SVM – Radial basis function -Gaussian (RBF kernel sklearn python package)
2. SVM – Support Vector Classification Linear (liblinear sklearn python package)
3. SVM – Support Vector Classification (libsvm sklearn python package)

2.5 LOGISTIC REGRESSION

It is a supervised binary classification algorithm that describes the relationship between one dependent binary variable and one or more nominal or ordinal independent variables.

Parameters

1. Penalty → Used for specifying the norm used in the penalization
2. C→ Inverse of regularization strength. Smaller Values are better

3. EXPREMENTAL SETUP

3.1 DATASET DESCRIPTION

The dataset provided by Porto Seguro contains confidential information and hence the complete description of attributes is not provided along with the dataset. The high-level description of the dataset features are as follows.

Training data instances 593212 with 59 attributes.

Testing data instances 18000 with 58 attributes (No class attribute).

Feature Type	Feature Count
Binary	17
Continuous	10
Categorical – Nominal	14
Categorical – Ordinal	16
Target - Target class field	1
Id	1

Table 1. Feature type and its count

Feature Name	Feature Description
ps_ind_18_bin, ps_calc_15_bin, ps_calc_17_bin, …	Binary features
ps_reg_03, ps_car_12, ps_calc_02, ….	Continuous features
ps_ind_05_cat, ps_car_02_cat, ps_car_10_cat, ….	Nominal type
ps_ind_15, ps_car_11, ps_calc_09, …	Ordinal type

Table 2. Feature Name and Description

Feature Name	Feature Category
ps_**ind**_14, ps_**ind**_13_bin, ps_**ind**_01, ….	Customer information
ps_**reg**_01, ps_**reg**_02, ps_**reg**_03, ……	Region information
ps_**car**_05_cat, ps_**car**_06_cat, ps_**car**_07_cat, ….	Car information
ps_**calc**_01, ps_**calc**_02, ps_**calc**_03, …..	Calculated fields

Table 3. Feature Name and Category

3.2 DATA EXPLORATION

The given dataset is initially analyzed to understand and identify suitable preprocessing techniques. In this task, the main characteristics of the data are summarized.

3.2.1 METADATA CREATION

A metadata dictionary is defined that contains information about variables.

Key	Value
Feature Name	Feature names
Role	Target/id/input
Type	Feature category
Required	True/False (if used in prediction or not)
Datatype	datatype

Table 4. Metadata of the attributes

3.2.2 TARGET FEATURE IMBALANCE

The dataset was found to have a major class imbalance which was identified by plotting the class distribution.

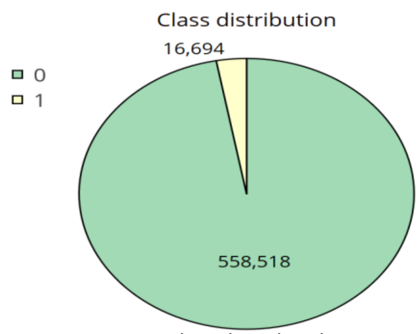

Figure 1. Class data distribution

3.2.3 INPUT FEATURE IMBALANCE

An analysis was made on the binary features to identify signs of potential imbalance.

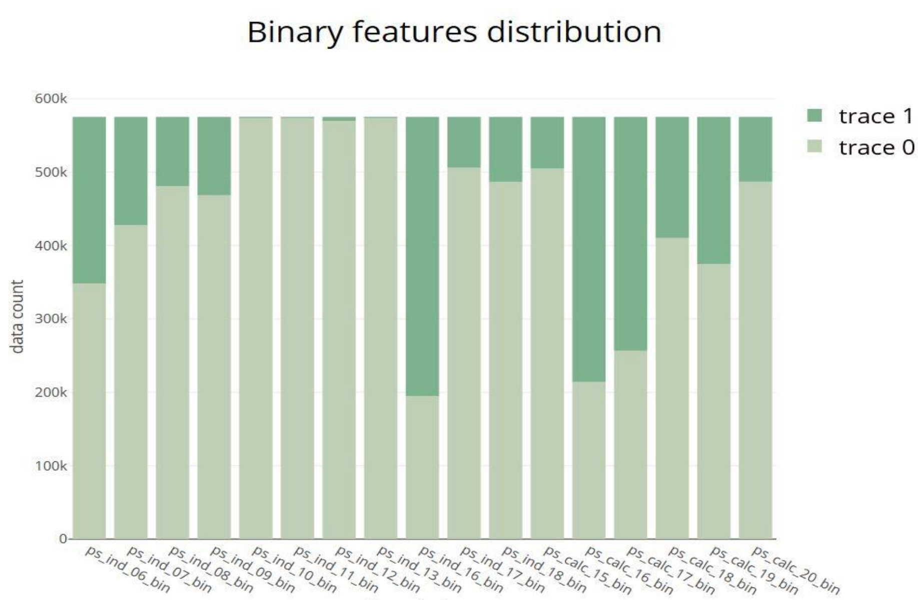

Figure 2. Binary features distribution

3.2.4 CORRELATION

Identifying correlation between features helps in understanding redundancy amongst them. If two features behave similarly or contribute the same level towards the target class, then they are considered as redundant features. One such feature can be retained, and the rest can be removed from prediction model. Heat map was plotted to analyze the same.

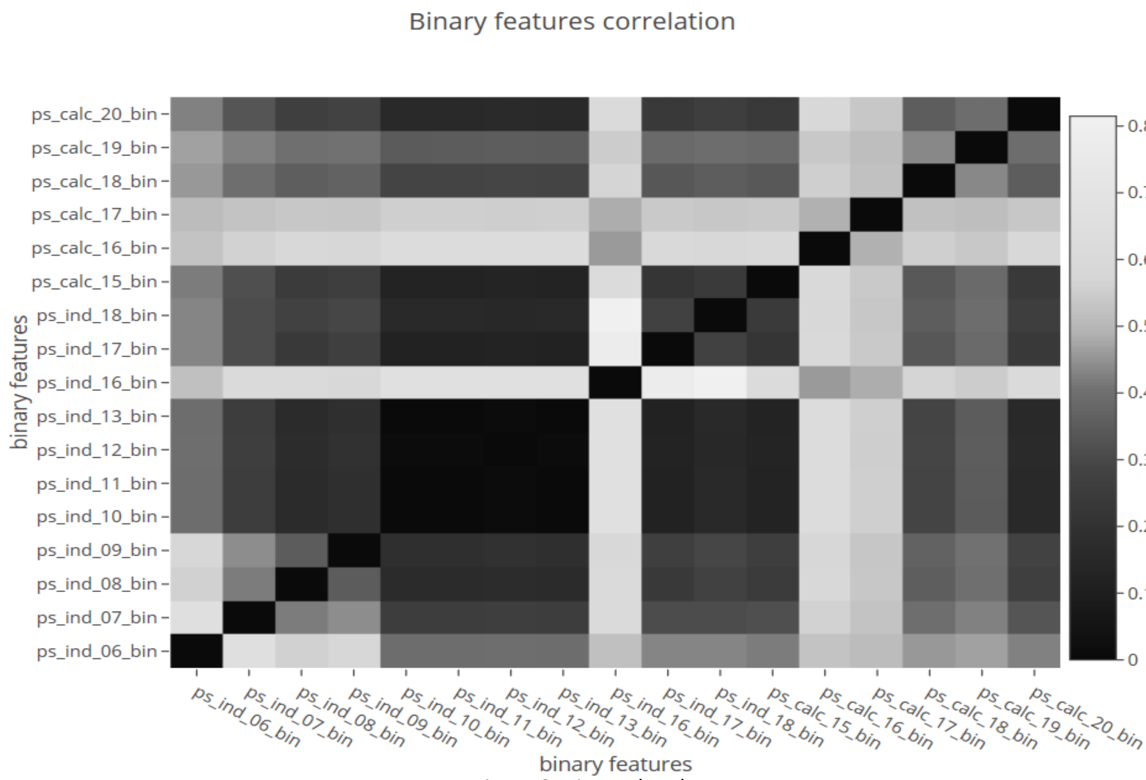

Figure 3. Binary data heat map

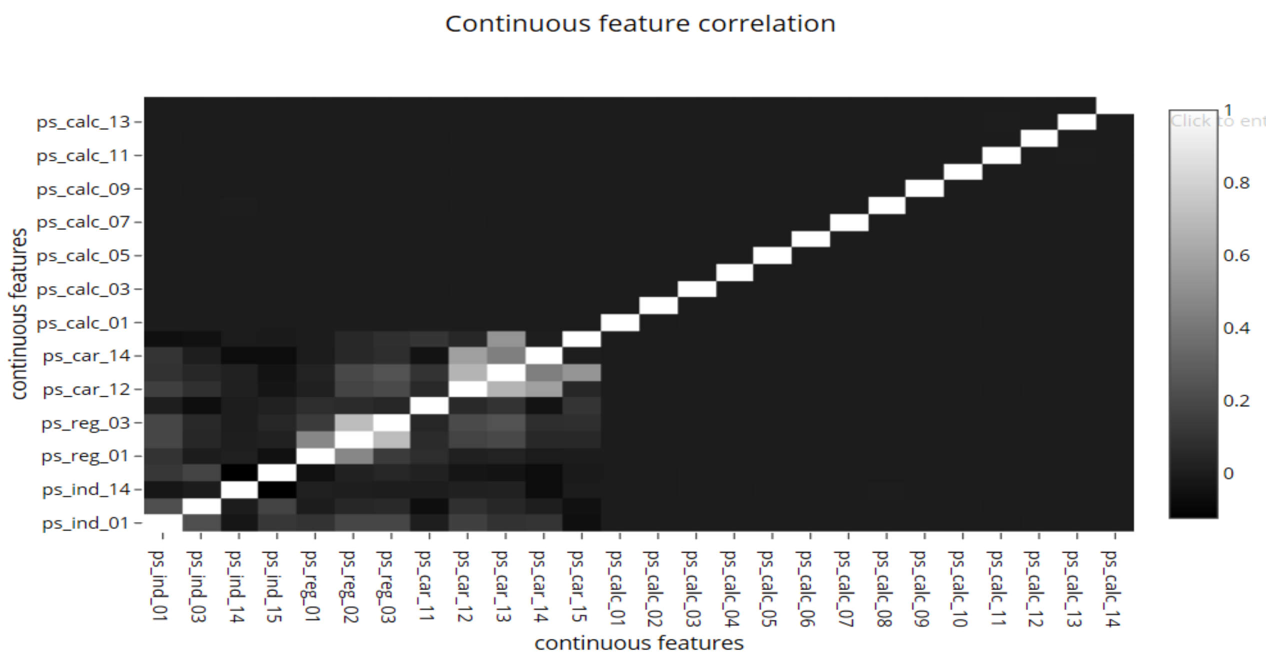

Figure 4. Continuous data heat map

3.2.5 PREDICTION POWER OF FEATURES

The correlation between the binary features and the target variable as well as continuous features with target variable is analyzed to understand the predictive power of the features. Those with minimum influence on the target field can be excluded from the prediction.

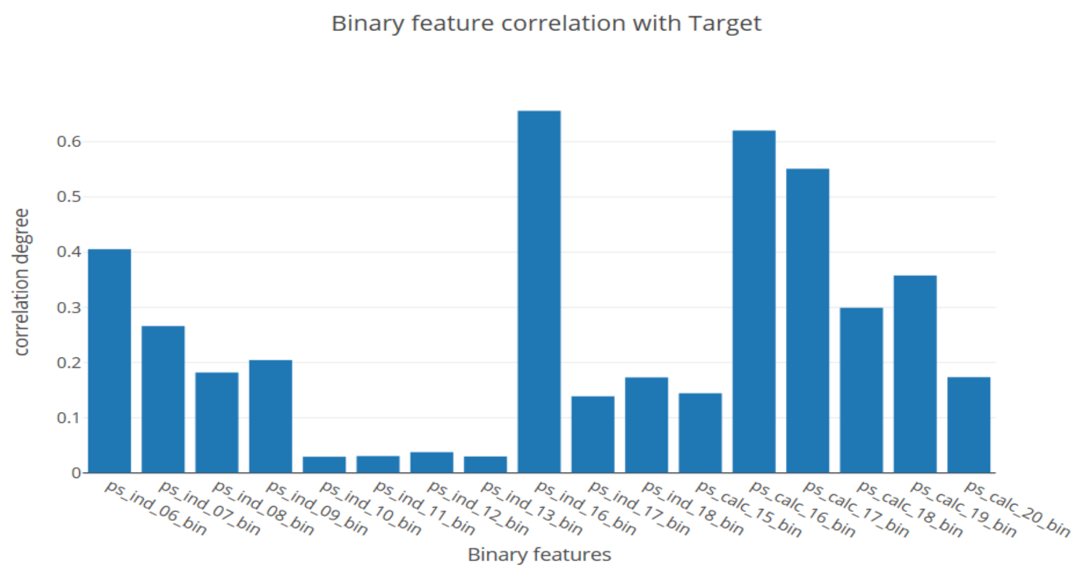

Figure 5. Binary feature correlation with Target

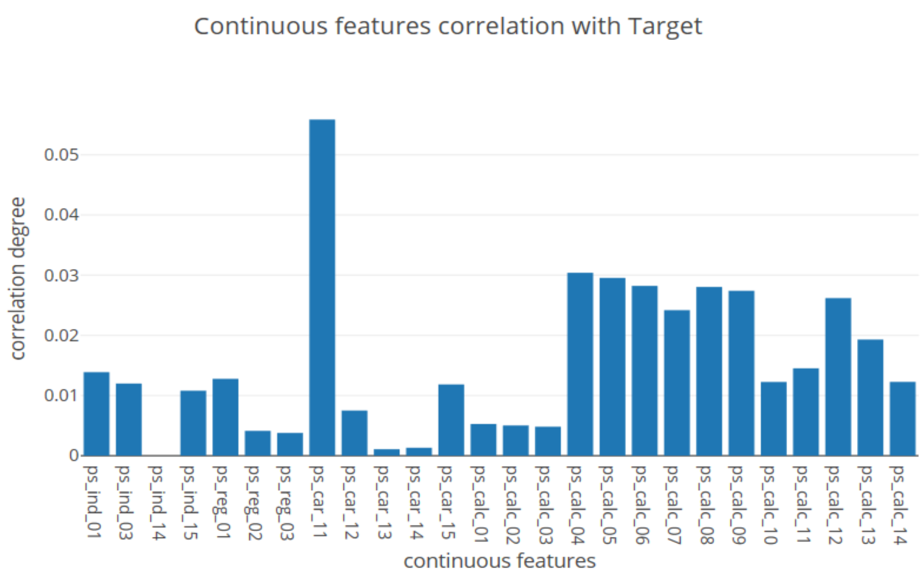

Figure 6. Continuous feature correlation with Target

3.2.6 MISSING VALUES

Missing values heavily influence the dataset quality. Features with high missing value counts can be removed from the prediction feature list. ps_car_03_cat and ps_car_05_cat was identified to have significant amount of missing values.

```
Variable ps_ind_02_cat has 203 records (0.04%) with missing values
Variable ps_ind_04_cat has 72 records (0.01%) with missing values
Variable ps_ind_05_cat has 5563 records (0.97%) with missing values
Variable ps_reg_03 has 104342 records (18.14%) with missing values
Variable ps_car_01_cat has 97 records (0.02%) with missing values
Variable ps_car_02_cat has 5 records (0.00%) with missing values
Variable ps_car_03_cat has 397725 records (69.14%) with missing values
Variable ps_car_05_cat has 257895 records (44.83%) with missing values
Variable ps_car_07_cat has 11026 records (1.92%) with missing values
Variable ps_car_09_cat has 535 records (0.09%) with missing values
Variable ps_car_11 has 5 records (0.00%) with missing values
Variable ps_car_12 has 1 records (0.00%) with missing values
Variable ps_car_14 has 41188 records (7.16%) with missing values
In total, there are 13 variables with missing values
```

Figure 7. Missing data values in percentage

3.3 DATA PREPROCESSING

To handle class imbalance, missing values, anonymity in data and multiple datatypes, the following pre-processing techniques were implemented.

3.3.1 FEATURE ELIMINATION

The features with too many missing values (approximately 50%) were eliminated because these features do not contribute towards classification. ps_car_03_cat and ps_car_05_cat was identified to have significant amount of missing values and was removed.

The remaining missing values were handled based on their datatypes. The Mean imputation was performed on float data types and the mode imputation was performed on integer datatypes.

Feature Name	Role	Level	Datatype	Imputer Type
ps_reg_03	Input	Interval	Float64	Mean
ps_car_12	Input	Interval	Float64	Mean
ps_car_14	Input	Interval	Float64	Mean
ps_car_11	Input	Ordinal	Int64	Mode
ps_ind_02_cat	Input	Nominal	Int64	Mode
ps_ind_04_cat	Input	Nominal	Int64	Mode
ps_ind_05_cat	Input	Nominal	Int64	Mode
ps_car_01_cat	Input	Nominal	Int64	Mode
ps_car_02_cat	Input	Nominal	Int64	Mode
ps_car_07_cat	Input	Nominal	Int64	Mode
ps_car_09_cat	Input	Nominal	Int64	Mode

Table 5. Features Vs Imputer type

3.3.2 ENCODING CATEGORICAL FEATURES

The categorical features are numbers substituted in the place of text. They do not hold any inherent order. Thus, Bayesian encoding technique was used to inject ordering [9]. The count of distinct values was taken as prior probability and passed into the gaussian function along with the smoothing factor to obtain a smoothing value. Encoding was performed by taking log-likelihood on the prior and the smoothing value. The above step was carried out inside k-fold cross validation.

Feature Name	Initial Range	Encoded Range
ps_ind_02_cat	1 to 4	0 to 1
ps_ind_04_cat	0 or 1	0 to 1
ps_ind_05_cat	0 to 6	0 to 1
ps_car_01_cat	0 to 11	0 to 1
ps_car_02_cat	0 or 1	0 to 1
ps_car_04_cat	0 to 9	0 to 1
ps_car_06_cat	0 to 17	0 to 1
ps_car_07_cat	0 or 1	0 to 1
ps_car_08_cat	0 or 1	0 to 1
ps_car_09_cat	0 to 4	0 to 1
ps_car_10_cat	0 to 2	0 to 1
ps_car_11_cat	1 to 104	0 to 1

Table 6. Features Vs Encoded Range

3.3.3 DATA STANDARDIZATION

The data standardization helps to make the source data consistent. Minmax Scalar is used in the proposed approach to transform each feature to a specified range (i.e. between 0 and 1).

3.3.4 DATA SAMPLING

To address class imbalance, two different approaches were used

3.3.4.1 UP-SAMPLING

To balance the class label count, up sampling was performed by duplicating the entries of minority class.

3.3.4.2 DOWN-SAMPLING

To balance the class label count, down sampling was performed by taking subset of the entries from the majority class.

4. EXPERIMENTAL RESULTS

4.1 CLASSIFIER ANALYSIS AND PARAMETER TUNING

In the proposed solution, the following classifiers were selected based on their compatibility with large datasets, efficiency, training speed and their ability to produce better results. The K-fold stratified cross validation was performed to preserve the percentage of data instances for each class.

4.1.1 RANDOM FOREST CLASSIFIER

10 - fold stratified cross validation was performed. F1_measure, macro averaged recall and the accuracy score was computed and tabulated for each fold.

The accuracy was plotted by computing the average for each of the three parameter sets. It was observed that the following parameter values gave better results for both validation data and test data.

n-tree=100, oob_score=True, random_state=13, n_jobs = -1, min_samples_leaf = 100

```
RF_model_cat1= RandomForestClassifier(100,oob_score=True,random_state=13)
RF_model_cat2= RandomForestClassifier(50,oob_score=True,random_state=13,n_jobs = -1)
RF_model_cat3= RandomForestClassifier(20,oob_score=True,random_state=13,n_jobs = -1,
                                      min_samples_leaf = 100)
```

Figure 8. Random Forest Parameter Values

Parameter Set 1= 100 trees, oob_score=True, random_state=13			
Fold Number	**Validation Data Results**		
	F1-Measure	**Recall**	**Accuracy**
1	0.477	0.501	0.90008984726
2	0.475	0.501	0.899610661875
3	0.477	0.502	0.900269541779
4	0.476	0.501	0.89991015274
5	0.474	0.500	0.899724451899
6	0.475	0.500	0.899904157182
7	0.475	0.500	0.899778350207
8	0.476	0.501	0.900137782304
9	0.476	0.501	0.900017971605
10	0.474	0.500	0.899838255556

Table 7. Performance results for Random forest parameter set 1

Parameter Set 2 = 50 Trees, oob_score=True, n_jobs=-1			
Fold Number	**Validation Data Results**		
	F1-Measure	**Recall**	**Accuracy**
1	0.476	0.501	0.899850254567
2	0.475	0.501	0.899550763702
3	0.477	0.502	0.900029949087
4	0.477	0.502	0.90008984726
5	0.477	0.502	0.899904157182
6	0.477	0.501	0.899604648377
7	0.475	0.501	0.899718444857
8	0.476	0.501	0.900077876954
9	0.476	0.501	0.89929910741
10	0.475	0.500	0.899598634158

Table 8. Performance results for Random forest parameter set 2

Parameter Set 3 = 20 Trees, n_jobs = -1, min_samples_leaf = 100			
Fold Number	Validation Data Results		
	F1-Measure	Recall	Accuracy
1	0.474	0.500	0.899970050913
2	0.474	0.500	0.899970050913
3	0.474	0.500	0.899970050913
4	0.474	0.500	0.899970050913
5	0.474	0.500	0.900023960704
6	0.474	0.500	0.900023960704
7	0.474	0.500	0.900017971605
8	0.474	0.500	0.900017971605
9	0.474	0.500	0.900017971605
10	0.474	0.500	0.900017971605

Table 9. Performance results for Random forest parameter set 3

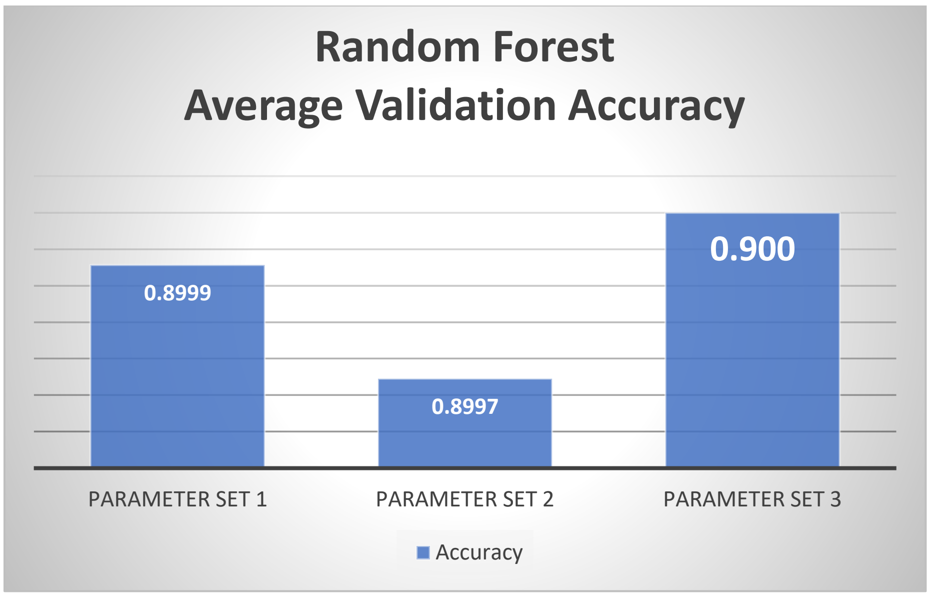

Figure 9. Validation Performance Graph for Random forest

Parameter set	Test Data Results		
	F1-Measure	Recall	Accuracy
1	0.456	0.501	0.833277777778
2	0.459	0.501	0.832166666667
3	0.455	0.500	0.833333333333

Table 10. Test Performance results for Random forest

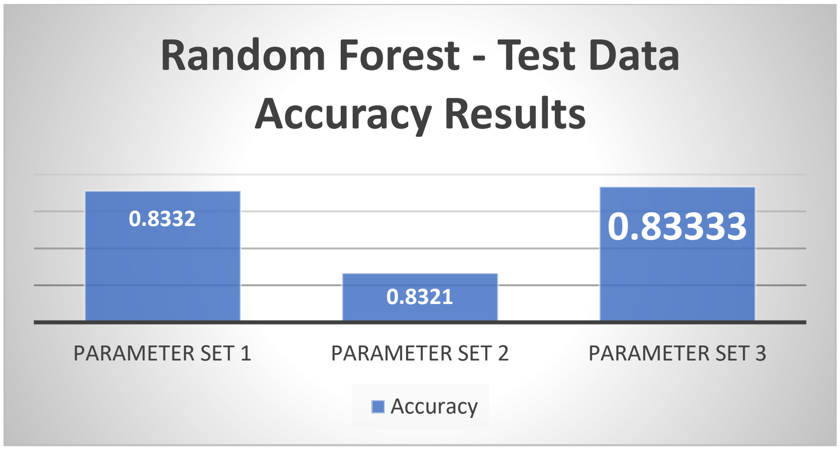

Figure 10. Test Performance Graph for Random forest

4.1.2 LOGISTIC REGRESSION

10 - fold stratified cross validation was performed. F1_measure, macro averaged recall and the accuracy score was computed and tabulated for each fold.

The accuracy was plotted by computing the average for each of the three parameter sets. It was observed that the following parameter values gave better results for both validation data and test data.

penalty=”l2”, and C =0.5

```
model_cat_1 = LogisticRegression(penalty='l2',C=1)
model_cat_2 = LogisticRegression(penalty='l1',C=2)
model_cat_3 = LogisticRegression(penalty='l2',C=0.5)
```

Figure 11. Logistic Regression Parameter Values

Parameter Set 1 = penalty='l2',C=1			
Fold Number	**Validation Data Results**		
	F1-Measure	**Recall**	**Accuracy**
1	0.497	0.500	0.987541179994
2	0.497	0.500	0.988559448937
3	0.497	0.500	0.989877208745
4	0.497	0.500	0.98939802336
5	0.497	0.500	0.988438960105
6	0.497	0.500	0.988918174194
7	0.497	0.500	0.988198646139
8	0.497	0.500	0.988857604984
9	0.497	0.500	0.989516563829
10	0.497	0.500	0.988617983586

Table 11. Performance results for Logistic Regression parameter set 1

Parameter Set 2= penalty='l1',C=2			
Fold Number	Validation Data Results		
	F1-Measure	Recall	Accuracy
1	0.497	0.500	0.988259958071
2	0.497	0.500	0.98957771788
3	0.497	0.500	0.989637616053
4	0.497	0.500	0.98766097634
5	0.497	0.500	0.989217683
6	0.497	0.500	0.987899844255
7	0.497	0.500	0.989157131732
8	0.497	0.500	0.989756185227
9	0.497	0.500	0.988318456838
10	0.497	0.500	0.988438267537

Table 12. Performance results for Logistic Regression parameter set 2

Parameter Set 3= penalty='l2',C=0.5			
Fold Number	Validation Data Results		
	F1-Measure	Recall	Accuracy
1	0.497	0.500	0.988140161725
2	0.497	0.500	0.988739143456
3	0.497	0.500	0.989457921533
4	0.497	0.500	0.988799041629
5	0.497	0.500	0.987899844255
6	0.497	0.500	0.98867856715
7	0.497	0.500	0.988857604984
8	0.497	0.500	0.988378362188
9	0.497	0.500	0.989336847781
10	0.497	0.500	0.989636374528

Table 13. Performance results for Logistic Regression parameter set 3

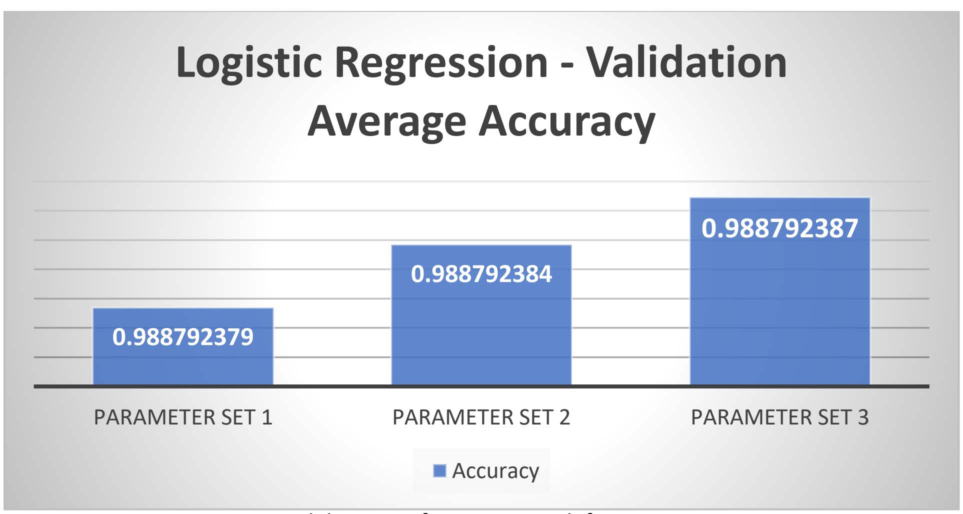

Figure 12. Validation Performance Graph for Logistic Regression

Parameter set	Test Data Results		
	F1-Measure	Recall	Accuracy
1	0.455	0.5	0.833333333333
2	0.455	0.5	0.833055555556
3	0.455	0.5	0.833353333333

Table 14. Test Performance results for Logistic Regression

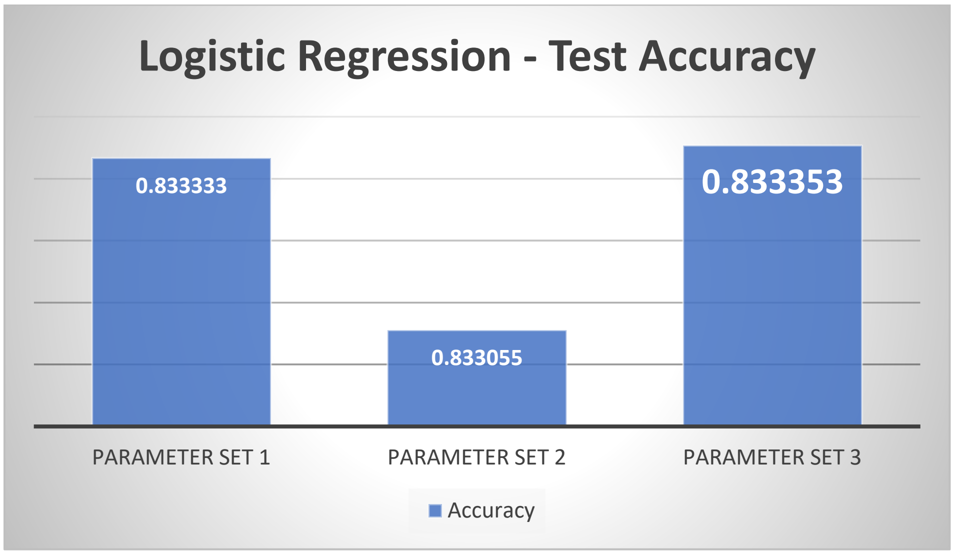

Figure 13. Test Performance Graph for Logistic Regression

4.1.3 LIGHT GRADIENT BOOST ALGORITHM

10 - fold stratified cross validation was performed. F1_measure, macro averaged recall and the accuracy score was computed and tabulated for each fold.

The Gini index was plotted by computing the average for each of the three parameter sets. Since Gini index is a measure of impurity, lower values are better [5]. It was observed that the following parameter values gave better results for both validation data and test data.

'max_depth': 3,'learning_rate': 0.05,'bagging_freq': 10,'min_split_gain': 0.5

Parameter Set 1 ='max_depth': 3,'learning_rate': 0.05, 'feature_fraction': 1, 'bagging_fraction': 1,'bagging_freq': 10,'min_split_gain': 0.5	
Fold Number	Validation Gini Index
1	0.00242316104108
2	0.0104107791041
3	0.024386387712
4	0.0444548689479
5	0.0702955241594
6	0.100272830247
7	0.136760390016
8	0.177340528222

9	0.226523869312
10	0.280248478358

Table 15. Performance results for Light Gradient Boost parameter set 1

Parameter Set 2= 'max_depth': 4,'learning_rate': 0.1 'feature_fraction': 0.9,'bagging_fraction': 0.9,'bagging_freq': 2,'min_split_gain': 0.1	
Fold Number	**Validation Gini Index**
1	0.0024595013084
2	0.0104300244853
3	0.0244749294215
4	0.0445835076147
5	0.0707545143066
6	0.10125016146
7	0.138054555619
8	0.178925269028
9	0.228428309188
10	0.282631749437

Table 16. Performance results for Light Gradient Boost parameter set 2

Parameter Set 3= 'max_depth': 5, 'learning_rate': 0.05, 'feature_fraction': 0.3,'bagging_fraction': 0.7, 'bagging_freq': 10	
Fold #	**Validation Gini Index**
1	0.00248263529866
2	0.0106579518672
3	0.0248758582392
4	0.0453254491174
5	0.0718228724553
6	0.102539057057
7	0.139871948761
8	0.18080173252
9	0.231227674521
10	0.286084924477

Table 17. Performance results for Light Gradient Boost parameter set 3

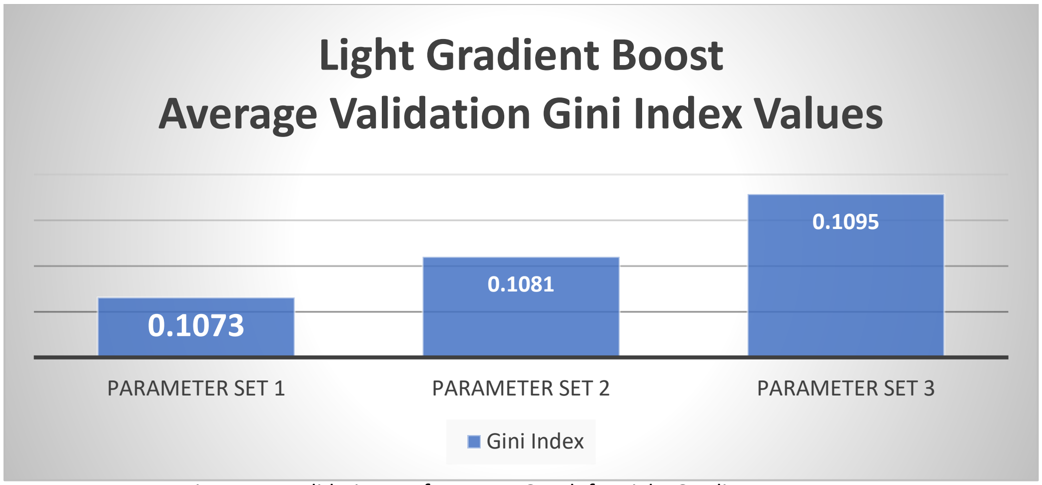

Figure 14. Validation Performance Graph for Light Gradient Boost

Parameter set	Test Results
	Gini Index
1	0.261606444444
2	0.264029066667
3	0.265844355556

Table 18. Test Performance results for Light Gradient Boost

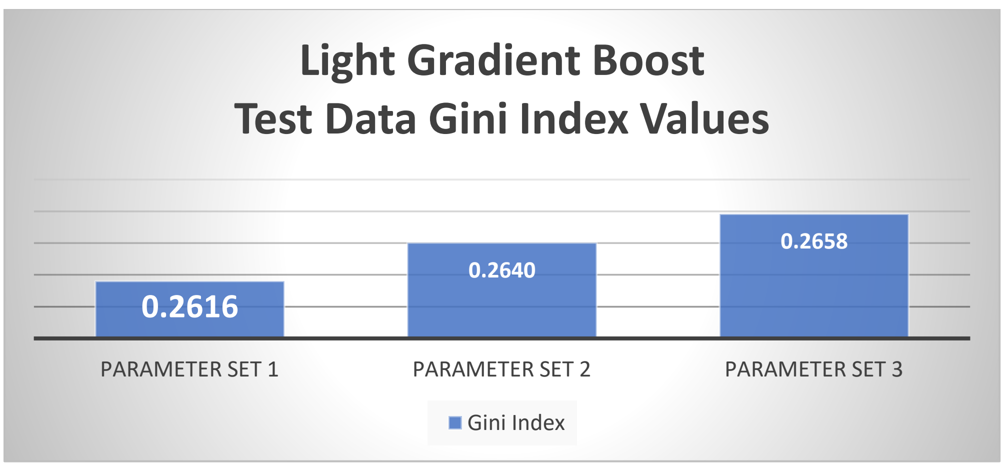

Figure 15. Test Performance Graph for Light Gradient Boost

4.1.4 SUPPORT VECTOR MACHINE CLASSFIERS

10 - fold stratified cross validation was performed. F1_measure, macro averaged recall and the accuracy score was computed and tabulated for each fold. Three different SVM classifiers was trained to check the classification accuracy.

SVM – RBF KERNEL			
Fold Number	Validation Data Results		
	F1-Measure	Recall	Accuracy
1	0.474	0.500	0.900
2	0.474	0.500	0.900
3	0.474	0.500	0.900
4	0.474	0.500	0.900

5	0.474	0.500	0.900
6	0.474	0.500	0.900
7	0.474	0.500	0.900
8	0.474	0.500	0.90019887
9	0.474	0.500	0.90020887
10	0.474	0.500	0.90078433

Table 19. Performance results for Support Vector Machine RBF Kernel

SVM – LINEAR KERNEL			
Fold Number	**Validation Data Results**		
	F1-Measure	**Recall**	**Accuracy**
1	0.474	0.500	0.900
2	0.474	0.500	0.900
3	0.474	0.500	0.900
4	0.474	0.500	0.900
5	0.474	0.500	0.900
6	0.474	0.500	0.900
7	0.474	0.500	0.900
8	0.474	0.500	0.90064899
9	0.474	0.500	0.90089432
10	0.474	0.500	0.90094367

Table 20. Performance results for Support Vector Machine Linear Kernel

SVM SVC LINEAR			
Fold Number	**Validation Data Results**		
	F1-Measure	**Recall**	**Accuracy**
1	0.474	0.500	0.900
2	0.474	0.500	0.900
3	0.474	0.500	0.900
4	0.475	0.500	0.900
5	0.475	0.500	0.900
6	0.475	0.500	0.900
7	0.475	0.500	0.900
8	0.475	0.500	0.900
9	0.475	0.500	0.900658932
10	0.475	0.500	0.900743289

Table 21. Performance results for Support Vector Machine SVC Linear

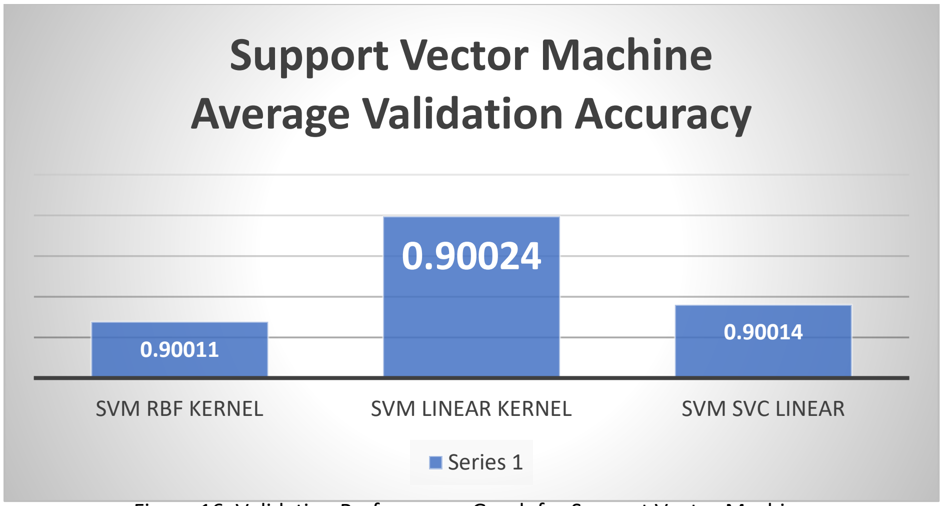

Figure 16. Validation Performance Graph for Support Vector Machine

Parameter set	Test Data Results		
	F1-Measure	Recall	Accuracy
1	0.455	0.500	0.832555555556
2	0.455	0.500	0.833335433333
3	0.460	0.502	0.833333333333

Table 22. Test Performance results for Support Vector Machine Classifiers

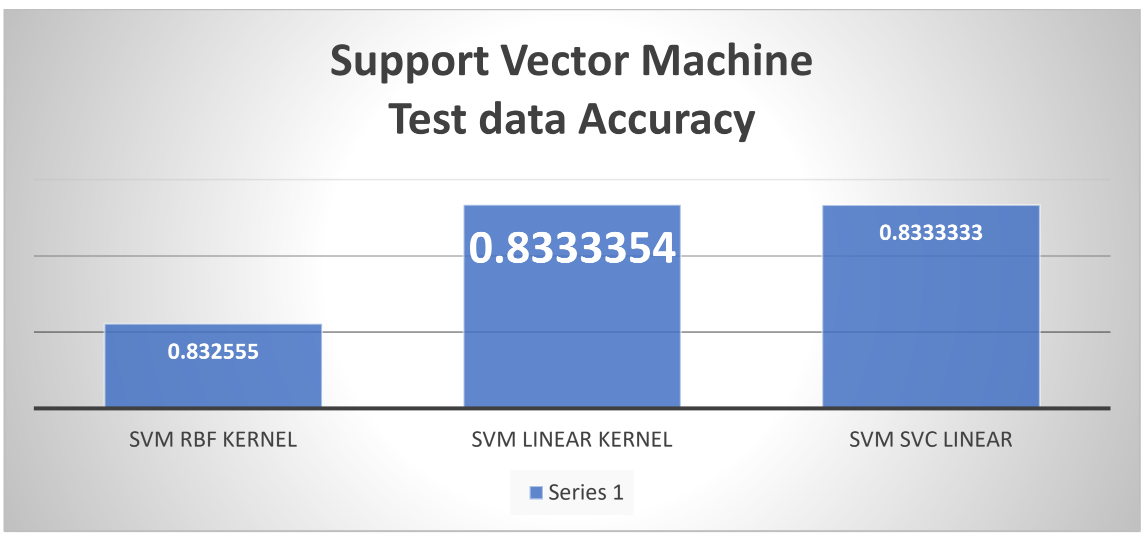

Figure 17. Test Performance Graph for Support Vector Machine

4.1.5 XTREME GRADIENT BOOST ALGORITHM

10 - fold stratified cross validation was performed. F1_measure, macro averaged recall and the accuracy score was computed and tabulated for each fold.

The Gini index was plotted by computing the average for each of the three parameter sets. Since Gini index is a measure of impurity, lower values are better [5]. It was observed that the following parameter values gave better results for both validation data and test data.

max_depth =15, eta= 0.5, silent=1, objective=binary:logistic, eval_metric=auc, learning_rate=0.1

Parameter Set 1= 'max_depth':7,'eta':1,'learning_rate':.05	
Fold Number	**Validation Gini Index**
1	0.00210226000579
2	0.00917708176428
3	0.0215412798231
4	0.0391949778431
5	0.0619558871796
6	0.0884218854683
7	0.121023141977
8	0.156599976142
9	0.199147714386
10	0.245947014762

Table 23. Performance results for Xtreme Gradient Boost parameter set 1

Parameter Set 2= 'max_depth':10, 'eta':0.8,'learning_rate':.01	
Fold Number	**Validation Gini Index**
1	0.00225997717136
2	0.00955110499383
3	0.0224144270516
4	0.041394496083
5	0.0660448530712
6	0.0940022021636
7	0.128297000416
8	0.166064705151
9	0.21263802926
10	0.262449756574

Table 24. Performance results for Xtreme Gradient Boost parameter set 2

Parameter Set 3 = 'max_depth':15,'eta':0.5,'learning_rate':.1	
Fold Number	**Validation Gini Index**
1	0.0010099294837
2	0.00413730285629
3	0.00934308741474
4	0.0175051601682
5	0.0265170863556
6	0.0385962472363
7	0.0524935165185
8	0.0697902118699
9	0.0885879350918
10	0.107961884281

Table 25. Performance results for Xtreme Gradient Boost parameter set 3

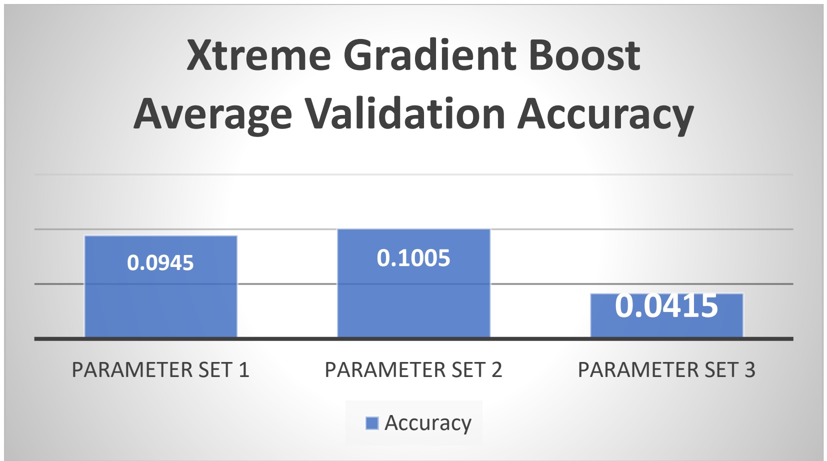

Figure 18. Validation Performance Graph for Xtreme Gradient Boost

Parameter set	Test Results
	Gini Index
1	0.223462266667
2	0.238775688889
3	0.0958367555556

Table 26. Test Performance results for Xtreme Gradient Boost

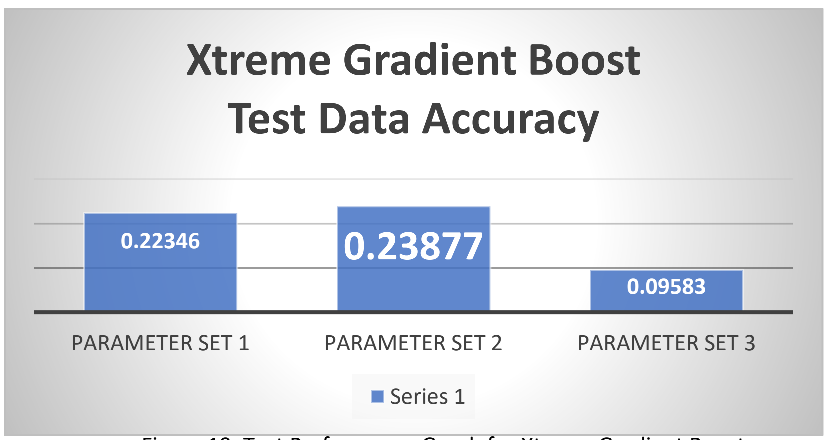

Figure 19. Test Performance Graph for Xtreme Gradient Boost

4.2 TIME AND MEMORY CONSUMPTION

Classifier Name	Memory Used (GB)	Running Time (Seconds)
Light Gradient Boost Algorithm	1.44	221
Random Forest Algorithm	1.03	45
Logistic Regression Algorithm	0.814	50
Xtreme Gradient Boost Algorithm	0.596	2485
SVM – Radial Basic Function Kernel	0.2788	2584
SVM – Linear Kernel	0.267	742
SVM -SVC Linear	0.246	28

Table 27. Memory and Time consumption for Various Implemented Algorithm

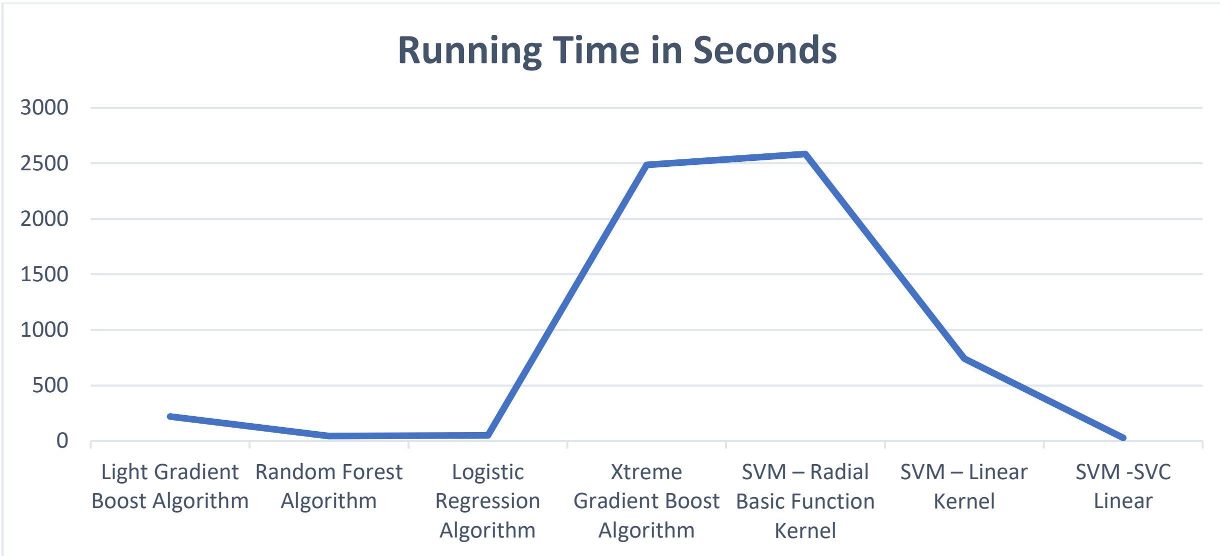

Figure 20. Running Time Graph

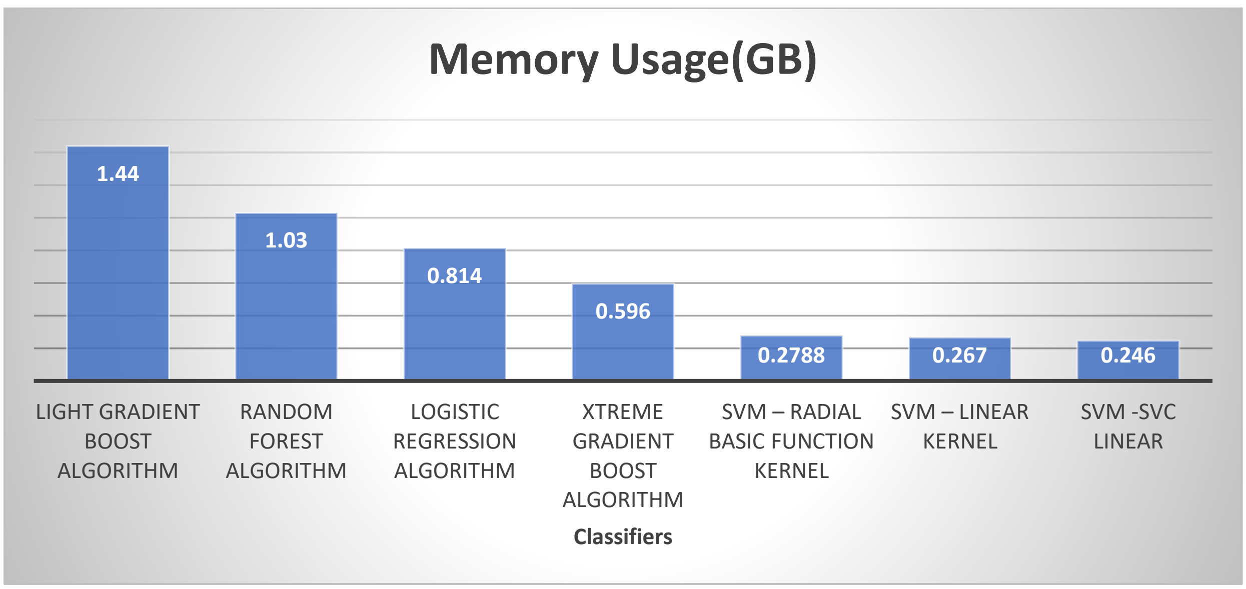

Figure 21. Memory Usage Graph

5. FUTURE WORK

Missing values can be handled sophisticatedly by using importance sampling on the incomplete data set. This involves using approximation of predictive distribution over the unobserved part of the data. Better Up sampling techniques like generating synthetic samples can be used instead of duplicating entries of the minority class. This technique selects random instances based on distance measure and alters one attribute at a time by a random amount with respect to its neighboring instance. Many such advancements in pre-processing techniques can be used to improve the performance of classification algorithms.

6. CONCLUSION

In this project, it was observed that all the implemented classifiers displayed a minor degree of variation in terms of accuracy. Logistic regression outperformed the other classifiers. In terms of memory usage and running time support vector machine svc linear was found to be efficient. Compared to light gradient descent algorithm, Xtreme gradient boosting had lower Gini index value and in turn proved to have low degree of impurity. Compare to other classifiers, Random forest algorithm was found to be costly in terms of memory usage.

7. REFERENCES

[1] Frempong, N.K., Nicholas, Nimo, Boateng, M.A., Decision Tree as a Predictive Modeling Tool for Auto Insurance Claims, International Journal of Statistics and Applications 2017, 7(2): 117-120

[2] Asma S. Alshamsi, Predicting Car Insurance Policies Using Random Forest, 978-1-4799-7212-8/14/$31.00 ©2014 IEEE

[3] Thorsten Joachims, A Support Vector Method for Multivariate Performance Measures, NSF awards IIS-0412894 and IIS-0412930

[4] Andy Liaw, Matthew Wiener, Classification and Regression by Random Forest, ISSN 1609-3631

[5] Comparison of Data Mining Techniques for Insurance Claim Prediction, [online], Available: http://www.ulb.ac.be/di/map/adalpozz/pdf/Claim_prediction.pdf

[6] Xiongfeng LI, Xinyu FAN, An SVM Based Analysis of US Dollar Strength

[7] Yaqi Li, Chun Yan, Wei Liu, Maozhen Li, Research and Application of Random Forest Model in Mining Automobile Insurance Fraud, 2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)

[8] Tianqi Chen, Carlos Guestrin. (2016). KDD '16, XGBoost: A Scalable Tree Boosting System

[9] Micci-Barreca, Daniele. (2001). A Preprocessing Scheme for High-Cardinality Categorical Attributes in Classification and Prediction Problems.. SIGKDD Explorations. 3. 27-32. 10.1145/507533.507538.