Course Project - IME692

Sajal Goyal(180651), Vasu Bansal(160776)

9 November, 2020

1 Introduction and Overview

We are given 2 files namely **train.csv** and **test.csv**. Given 20 features and the dependent variable **y**, we have to predict the class of binary variable **y**(0 or 1).

2 Visualization of the Dataset

The training dataset contains 200 rows. All the 20 variables are of Numeric type.

Class	Number of points
y=1	97
y=0	103

The above table shows that distribution of data among 2 classes is more or less same and there is no class imbalance. To get correlation between any two variables - X_1 and X_2, we know -

$$Corr(X_1, X_2) = \frac{\sum_N (x_{1i} - \bar{x}_1)(x_{2i} - \bar{x}_2)}{\sqrt{\sum_N (x_{1i} - \bar{x}_1)^2 \sum_N (x_{2i} - \bar{x}_2)^2}}$$

where N=200 and X_1 and X_2 can be any variable between x_1 to x_{20}. After getting the correlation matrix, we observe that variables are highly uncorrelated and all correlation values < 0.3 for any 2 variables.

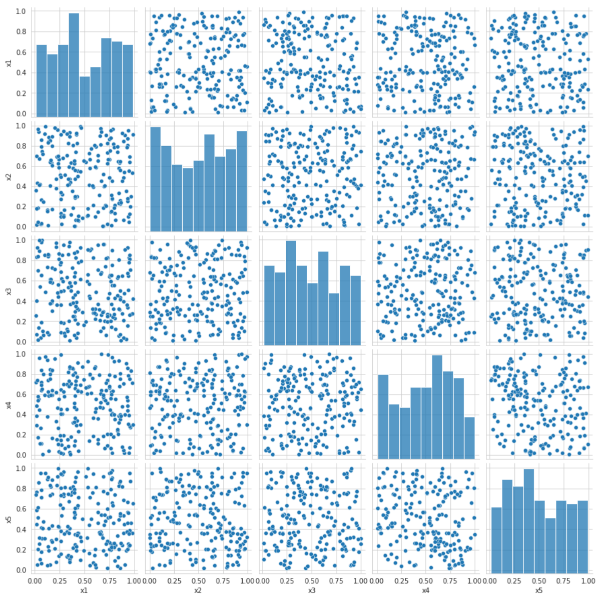


Figure 1: Pair plot of variables - x1 to x5. (Similar trend is observed for other variables)

3 Preprocessing

We have used **train_test_split** function from Scikit-learn library to split the training dataset into 70% train and 30% test set. Sklearn (or Scikit-learn) is a Python library that offers various features for data processing that can be used for classification, clustering, and model selection. Selecting a proper model allows us to generate accurate results when making a prediction. To do that, we need to train our model by using a specific dataset. Then, we validate the model against another dataset. As we cannot use test dataset to validate our model, we'll need to split the training dataset by using **train_test_split** function from Sklearn model selection.

4 Different Classification models

A classification model attempts to draw some conclusion from observed values. Given one or more inputs a classification model will try to predict the value of one or more outcomes. There are two approaches to machine learning: supervised and unsupervised. We will use supervised learning approach here.
There are lot of classification algorithms available now but it is not possible to conclude which one is superior to other. It depends on the application and nature of available data set. So we will now discuss different classification models.

4.1 Logistic Regression (Best Classifier in our case)

The logistic function is a sigmoid function, which takes any real input $t \in \mathbb{R}$ and outputs a value between zero and one for the logit, this is interpreted as taking input log-odds and having output probability. The standard logistic function $\sigma : \mathbb{R} \to (0, 1)$ is defined as follows:

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

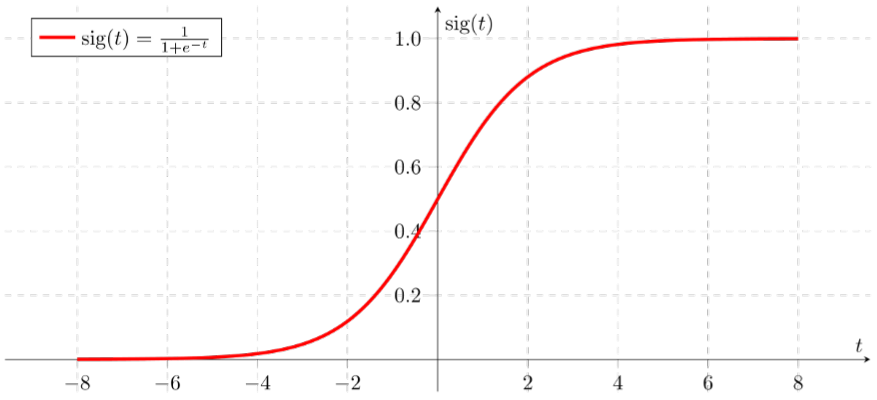


Figure 2: Sigmoid Function

After training the model on 70% of the train data, we get **70%** accuracy on the validation set and **81%** accracy on test set.
CONFUSION MATRIX for validation set:

	Predicted 0	Predicted 1
Actual 0	23	7
Actual 1	11	19

CONFUSION MATRIX for test set:

	Predicted 0	Predicted 1
Actual 0	409	82
Actual 1	109	400

We also performed Principal Component Analysis (PCA) to see if we can reduce the dimensionality of the input data by transforming a large set of variables into a smaller one that still contains most of the information in the large set. We observe that taking all 20 features gives the best accuracy on validation set which implies that all the features are important for the prediction.

n-components	Accuracy on validation set	n-components	Accuracy on validation set
3	0.5167	12	0.6333
4	0.55	13	0.6167
5	0.5167	14	0.55
6	0.5167	15	0.5333
7	0.5333	16	0.5333
8	0.5833	17	0.6833
9	0.6333	18	0.70
10	0.6167	19	0.7333
11	0.6167	20	0.7667

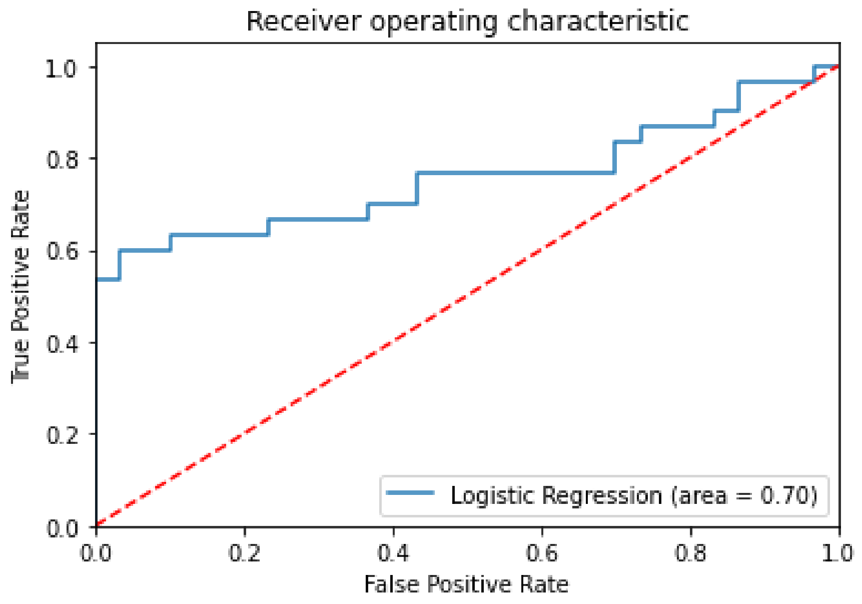


Figure 3: ROC curve

A Receiver Operator Characteristic (ROC) curve is a graphical plot used to show the diagnostic ability of binary classifiers. A ROC curve is constructed by plotting the true positive rate (TPR) against the false positive rate (FPR). The true positive rate is the proportion of observations that were correctly predicted to be positive out of all positive observations (TP/(TP + FN)). Similarly, the false positive rate is the proportion of observations that are incorrectly predicted to be positive out of all negative observations (FP/(TN + FP)).
The ROC curve shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR). Classifiers that give curves closer to the top-left corner indicate a better performance. In our case the ROC curve is closer to top left corner, meaning the true positive rate is greater than false positive rate, which is good for any classifier. As a baseline, a random classifier is expected to give points lying along the diagonal (FPR = TPR). The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

4.2 Decision Tree

Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. A tree

can be “learned” by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. Decision tree induction is a typical inductive approach to learn knowledge on classification. Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. An instance is classified by starting at the root node of the tree,testing the attribute specified by this node,then moving down the tree branch corresponding to the value of the attribute as shown in the above figure.This process is then repeated for the sub-tree rooted at the new node.
After implementing the **DecisionTreeClassifier** from sklearn library, we get a decision tree of depth 8. The below figure shows our decision tree model till depth 2.

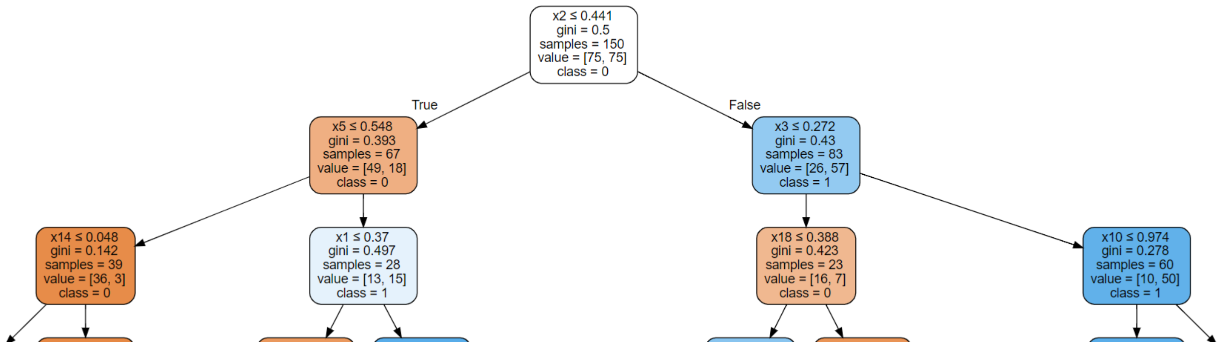


Figure 4: Just a glimpse of how the initial nodes would split

We got **58%** accuracy on validation set and **64.6%** accuracy on test set.
CONFUSION MATRIX for test set:

	Predicted 0	Predicted 1
Actual 0	301	190
Actual 1	164	345

4.3 k-Nearest Neighbors

k-nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions like euclidean distance). A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its k-nearest neighbors measured by a distance function. If **k** = 1, then the case is simply assigned to the class of its nearest neighbor.

In general, a large **k** value is more precise as it reduces the overall noise but there is no guarantee. Historically, the optimal **k** for most datasets has been between 3-10. To select the **k** that's right for our data, we run the kNN algorithm several times with different values of **k** and choose the **k** that reduces the number of errors we encounter while maintaining the algorithm's ability to accurately make predictions when it's given data it hasn't seen before.

no. of nearest neighbour	Accuracy on validation set	no. of nearest neighbour	Accuracy on validation set
1	0.62	5	0.5
2	0.62	6	0.62
3	0.56	7	0.58
4	0.66	8	0.6

From the above table, we observe that **k** = 4 gives the best accuracy. CONFUSION MATRIX for test set:

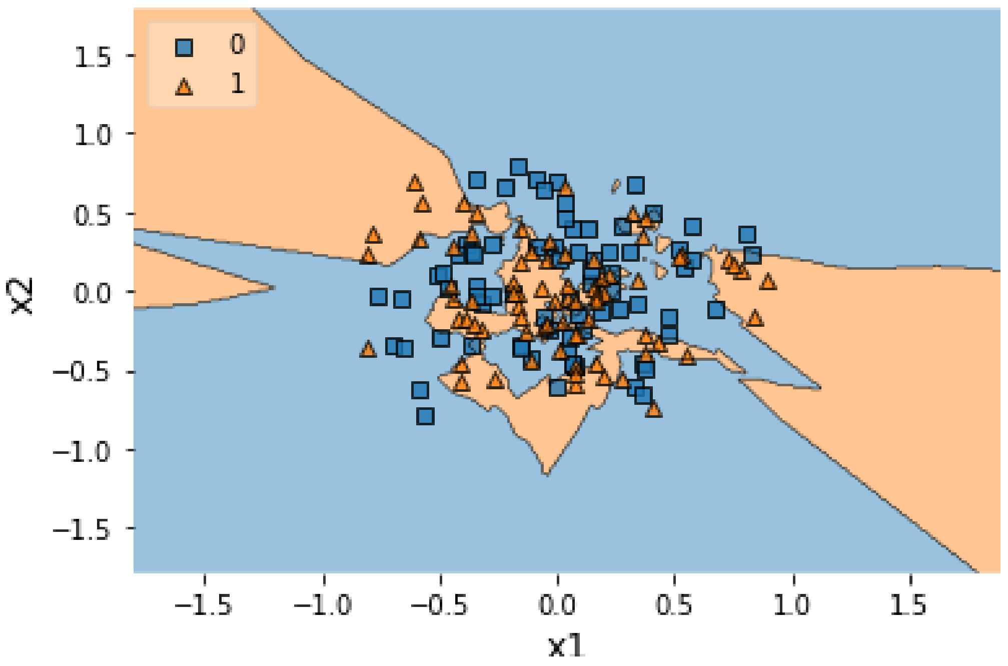


Figure 5: k-Nearest Neighbors decision boundary after applying PCA (n = 2 components)

	Predicted 0	Predicted 1
Actual 0	381	110
Actual 1	219	290

We get **66%** accuracy on validation set and **67%** accuracy for test set

4.4 Random Forest

A forest is comprised of trees. Generally more trees it has, the more robust a forest is. Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance. It

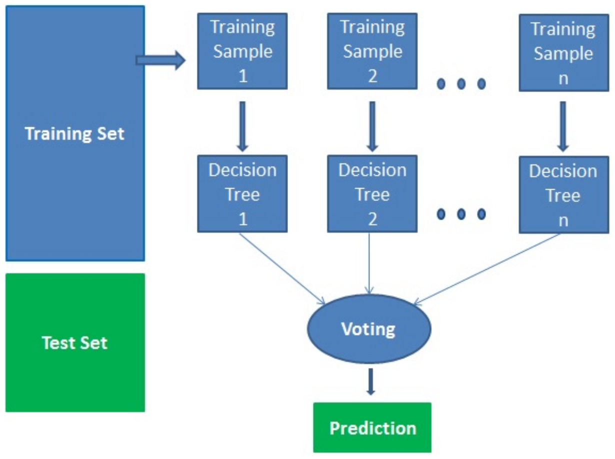


Figure 6: Random Forest general approach

technically is an ensemble method (based on the divide-and-conquer approach) of decision trees generated

on a randomly split dataset. The individual decision trees are generated using an attribute selection indicator such as information gain, gain ratio, and Gini index for each attribute. Each tree depends on an independent random sample. In a classification problem, each tree votes and the most popular class is chosen as the final result. In the case of regression, the average of all the tree outputs is considered as the final result. It is simpler and more powerful compared to the other non-linear classification algorithms.
We get **62%** accuracy on validation set and **81%** accuracy on test set.
CONFUSION MATRIX for test set:

	Predicted 0	Predicted 1
Actual 0	383	108
Actual 1	80	429

4.5 SVM

Support Vector Machine (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.

SVMs can be used on nonlinear data also using something called a Kernel trick. Basically it projects our data into a higher dimensional space such that our data can be separated by a hyperplane in this space.

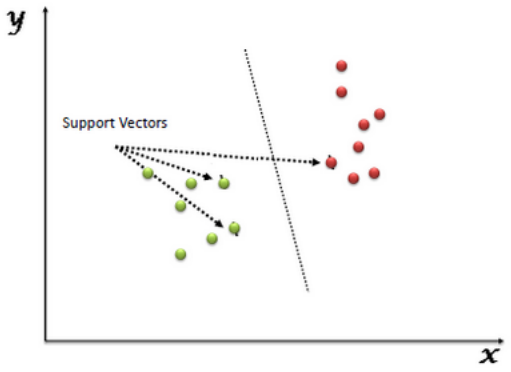


Figure 7: SVM classifier

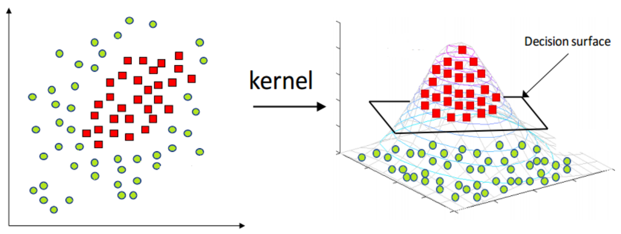


Figure 8: Kernel trick using Gaussian kernel

4.5.1 Parameters involved

Classifer object for SVM classifier is created using the below command

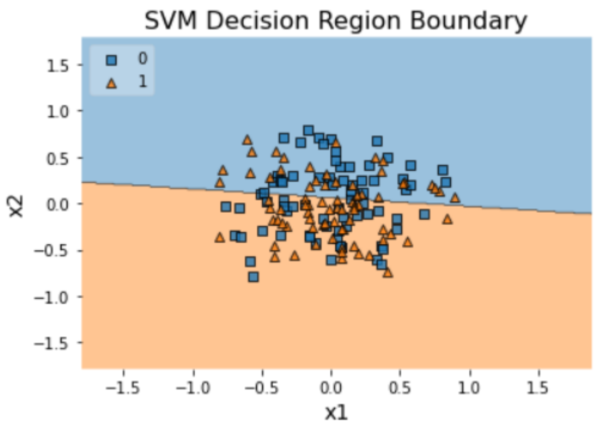


Figure 9: Decision boundary of SVM

```
    sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma=0.0, coef0=0.0, shrinking=True, \
    probability=False,tol=0.001, cache_size=200, class_weight=None, \
    verbose=False, max_iter=-1, random_state=None)
```

The major parameters which affect the performance of the classifer are parameters C, *kernel* and gamma(γ). We trained these parameters using gridsearch. For this we created a GridSearchCV() object with parameter combinations we wanted to test.

```
    # Applying grid search to find best model and best parameters
from sklearn.model_selection import GridSearchCV
parameters = [{'C':[1, 10, 100, 1000], 'kernel':['linear']},
              {'C':[1, 10, 100, 1000], 'kernel':['rbf'], 'gamma':[0.7, 0.71, 0.72, 0.73, 0.74]}]
grid_search = GridSearchCV(estimator = classifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv=10,
                           n_jobs=-1)
grid_search = grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
```

The best combination out of these was C = 0.1 and using a linear kernel and gamma = 0.71

4.6 XGBoost

XGBoost (eXtreme Gradient Boosting) is an advanced implementation of gradient boosting decision trees. Standard GBM implementation has no regularization like XGBoost, therefore it also helps to reduce overfitting. XGBoost uses parallel processing which makes it extremely fast compared to other ensembling techniques. Also its highly flexible and has inbuilt features for handling missing values in our data.
In gradient boosting our weak learners are high bias and low variance models. Gradient boosting decision trees uses decision trees having depth 1. These are also called decision stumps.

```
    # Applying grid search to find best model and best parameters
from sklearn.model_selection import GridSearchCV
param_test1 = {'max_depth':range(3,10,2), 'min_child_weight':range(1,6,2), \
'learning_rate': [0.01, 0.1, 1] }

classifier = XGBClassifier(learning_rate =0.1, n_estimators=140, \
```

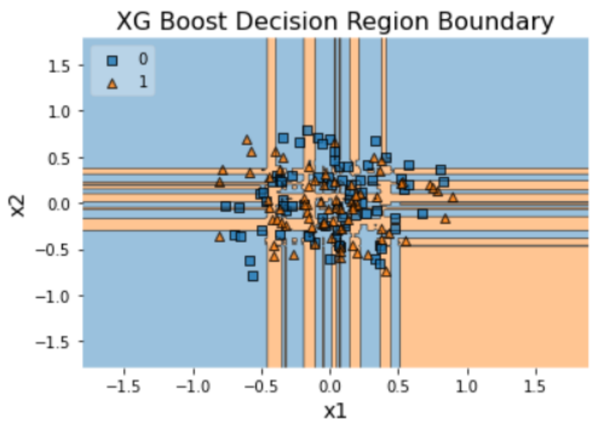
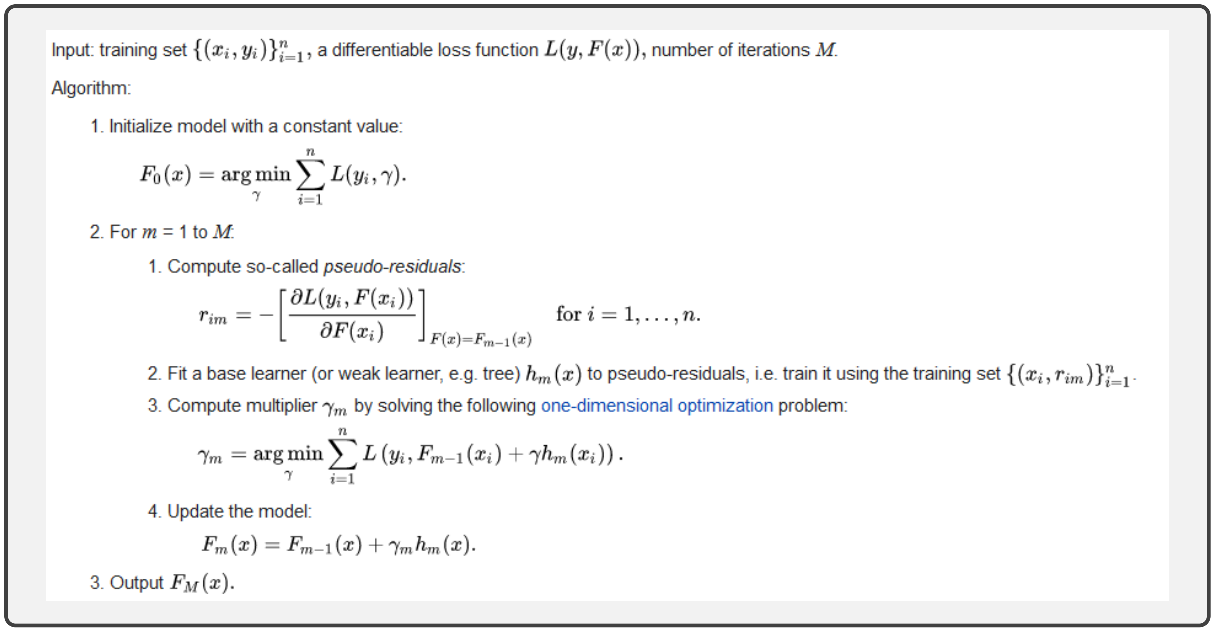


Figure 10: Decision boundary of XGBoost

```
max_depth=5, min_child_weight=1, gamma=0, subsample=0.8, \
colsample_bytree=0.8, objective= 'binary:logistic', nthread=4, \
scale_pos_weight=1, seed=27)
gsearch1 = GridSearchCV(estimator = classifier, param_grid = \
param_test1, scoring='roc_auc', n_jobs=4, iid=False, cv=5)
gsearch1.fit(X_train, y_train)
```



```
Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations $M$.
Algorithm:
  1. Initialize model with a constant value:
       $F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$
  2. For $m$ = 1 to $M$:
       1. Compute so-called pseudo-residuals:
            $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$   for $i = 1, \ldots, n.$
       2. Fit a base learner (or weak learner, e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
       3. Compute multiplier $\gamma_m$ by solving the following one-dimensional optimization problem:
            $\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$
       4. Update the model:
            $F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$
  3. Output $F_M(x)$.
```

We used grid search to find optimal parameters for the classifier. Most important hyper parameters are maximum depth and min_child_weight.

5 Conclusion

We observe that for **Logistic Regression**, we get maximum test accuracy as well as validation accuracy as compared to other models. It has misclassification error rate of **19.1%** on the test set which is less than **20%**. Logistic regression is intrinsically simple, it has low variance and so is less prone to over-fitting while other algorithms can be scaled up to be very complex, are more liable to over-fit. Logistic Regression is very robust to noise while other algorithms are not.

6 References

We mainly referred to various blogs on Analytics Vidhya and articles on Medium and Towards Datascience websites.

1. https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/
2. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-2
3. https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf3
4. https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/
5. https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-pytho
6. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwi32Y6jh_bsAh
7. https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/?utm_sour
8. https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html
9. https://towardsdatascience.com/the-complete-guide-to-decision-trees-28a4e3c7be14
10. https://www.displayr.com/what-is-a-roc-curve-how-to-interpret-it/
11. https://www.datacamp.com/community/tutorials/random-forests-classifier-python
12. https://www.saedsayad.com/k_nearest_neighbors.htm
13. https://builtin.com/data-science/step-step-explanation-principal-component-analysis

7 Contribution by each team member

1. **SAJAL GOYAL** (180651)
   - Visualizing and preprocessing part
   - Generating pair plot of training data set
   - Generating ROC curve and interpreting it in Logistic Regression
   - Generating confusion matrix for various models
   - Worked on Logistic Regression, Decision Tree, k-Nearest Neighbors
   - Doing PCA in some classifiers
2. **VASU BANSAL** (160776)
   - Writing theory for all the models
   - Applying Grid Search in SVM, XGBoost
   - Worked on Random Forest, SVM, XGBoost
   - Generating Decision Boundaries in kNN, SVM and XGBoost
   - Tweaking the parameters of code to improve accuracy