Customer Retention for the Google Merchandise Store

Group 10: James Diffenderfer, Anshika Saxena, Sanchit Deora, Jay Patel
Github Link: https://github.com/chrundle/ids-fall-2019-project-group10

Report Outline:

Description of the Problem and Data:

The problem and data was originally presented as a Kaggle challenge on predicting customer revenue (available here). We decided that we would use the data from this challenge to instead predict customer retention. Our goal is as follows: Given data collected on a Google Merchandise Store (GStore) customer, predict if that customer will return to shop at the GStore again. The data was originally provided pre-partitioned into training and testing sets of 25 GB and 8 GB, respectively. The training set consisted of user transactions between August 1, 2016 and April 30, 2018 while the testing set consisted of user transactions between May 1, 2018 and October 15, 2018. The original data set consists of the following 13 features: fullVisitorId, channelGrouping, date, sessionId, socialEngagementType, trafficSource, visitId, visitNumber, visitStartTime, **device**, **geoNetwork**, **totals**, and **hits**. The bold features were provided as JSON data and required extra preprocessing to extract additional features. Since the goal of our project was different from the original goal of the Kaggle challenge, we had to create and additional feature for each customer indicating whether or not they returned to shop at the GStore. A discussion is provided on this process in the following section.

Data Pre-processing:

a. **Subsampling and Creating Labels for Customer Retention**: [James]

The data set was pulled from a Google Analytics Customer Revenue Prediction Kaggle challenge designed to predict customer spending based on previous customer transaction data. Our goal for the project was to instead use this data to predict customer retention, that is, whether or not a customer will return based on their transaction data. As such, a key step in our data preprocessing pipeline was to generate a new feature, which we called ‘returningCustomer’, indicating whether or not a customer returned to shop at the Google Merchandise Store based on their transaction data. To achieve this goal, we made use of the ‘fullVisitorId’ feature which is a unique integer assigned to customers shopping at the Google Merchandise Store. If the customer returns again in the future their transaction data will have the same ‘fullVisitorId’ value as before. Hence, we assigned a value of 1 to the new ‘returningCustomer’ to customer’s whose ‘fullVisitorId’ appeared more than once in the training set and similarly for the testing set. If the ‘fullVisitorId’ appeared only once we assigned a value of 0 to the new ‘returningCustomer’ feature.

Another key step in the data preprocessing phase was to subsample the data set in a meaningful way. The original training set was approximately 25 GB and the original testing set was approximately 8 GB which are very large data sets to work with. To reduce the training and validation time for the model phase of the project, we decided to use subsampling to reduce the volume of our data set. The subsampling pipeline was designed to ensure that the same percentage of returning customers present in the original data set was present in the subsampled data set. To ensure this, before subsampling the data set was split into two groups: The first in which all of the entries had the value 0 in the ‘returningCustomer’ column and the second in which all of the entries had the value 1 in the ‘returningCustomer’ column. The data in each of these groups was then clustered over a subset of the features using KMeans clustering with a choice of five cluster centers. The choice of cluster centers was determined by running KMeans clustering with different numbers of cluster centers and using the elbow method.

Following the clustering, we used stratified sampling to select the same percentage of points from each of the clusters. Before subsampling, the full training and testing sets contained approximately 33.3% and 39.4% of returning customers, respectively. After the subsampling routine the subsampled training and testing sets contained approximately 33.4% and 39.4% of returning customers, respectively. Hence, our subsampled data sets preserved a key statistic from the original data sets. The purpose of the clustering used during the subsampled was to encourage the subsampling routine would preserve distribution statistics of other features present in the original data sets. After subsampling, the training set contained 1,537,503 samples and the testing set contained 361,429 samples which corresponds to a 81/19 training/testing split.

b. **Feature Engineering and Transformation**: [Anshika]

The dataset had many non-numeric columns with valuable information. Using Feature Engineering, such columns can be broken down to numeric values so that the information provided by the columns is preserved. Feature Transformation is also applied on numeric columns to represent the data in a manner that is easy for the model to understand and find patterns in. Feature Engineering and Transformation helps in improving the statistical understanding of the model.

- Date and Time (Cyclic Features) - The date feature in the dataset was co-joined and represented in the format [YMD] with no space in between like -- [20171016] representing 16th October,2017.This numeric column would not make it easier for the model to understand what the feature represents. For this reason, the column was split up into three new columns as - year , month ,date.

  Moreover, for normalising the features (so they don't end up far apart in the mapping) and to preserve the cyclic nature of the features, the sine and cosine values of the features were taken and stored in the respective columns. This way, Date related features will be mapped close to each other and may help the model understand them better.

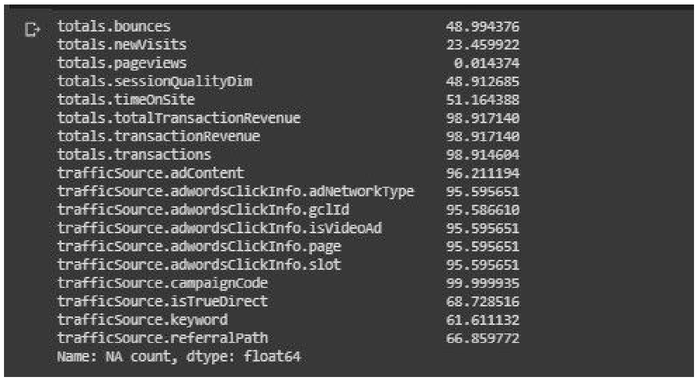
- Latitude and Longitude - The dataset has a column named -- [**geoNetwork.country]** -- which represented the country of the customer purchasing in the google store. It is a crucial information for a model trying to patterns which would be not as effectively represented in binning. For this reason, this column was converted to two numeric columns - Latitude and Longitude -- so that the notion of distance is preserved when a model looks at the order. The dataset called - countries.csv was taken from the US government record which had three columns - country_name , country_Latitude and country_Longitude. A left join was performed on the column country_name from the countries.csv and geoNetwork.country from the google dataset to add the Latitude and Longitude columns.

- Aggregations on the columns - An aggregation performed using the information in the features can help increase the statistical understanding of the model. For this reason, a new column called [country_dem] is added to the dataset. It takes into account the total number of orders coming from a country and thus helps the model to understand any relation that country regions might have with customer retention. For example, high number of country_dem for an order means that it’s more likely that the customer might visit the store again because of the high number of orders that come from the area that the customer belongs to. For this column, the columns were grouped by the country column and each group was aggregated to give the density/demand of orders from that country.

c. **Data Cleaning**: [Sanchit]

Given the size of the data set, it was forecasted that there would be many features with missing values. This phase of the pipeline was dedicated to dealing with those missing values. Cleaning these columns was instrumental to the pre-processing since these missing values hampered the ability of the model to classify the given data into class labels accurately. In addition, there were features in the data set with more than 90% of their data missing. These features were dropped from the data set which in-turn helped with improving the training time significantly.

The data cleaning process started with an iteration over each column in the data set. These columns were scanned for missing values. If a column had even a single missing value, a percentage was calculated for the missing values of that column. However, if the column had no missing values, the column was ignored.



```
totals.bounces                                48.994376
totals.newVisits                              23.459922
totals.pageviews                               0.014374
totals.sessionQualityDim                      48.912685
totals.timeOnSite                             51.164388
totals.totalTransactionRevenue                98.917140
totals.transactionRevenue                     98.917140
totals.transactions                           98.914604
trafficSource.adContent                       96.211194
trafficSource.adwordsClickInfo.adNetworkType  95.595651
trafficSource.adwordsClickInfo.gclId          95.586610
trafficSource.adwordsClickInfo.isVideoAd      95.595651
trafficSource.adwordsClickInfo.page           95.595651
trafficSource.adwordsClickInfo.slot           95.595651
trafficSource.campaignCode                    99.999935
trafficSource.isTrueDirect                    68.728516
trafficSource.keyword                         61.611132
trafficSource.referralPath                    66.859772
Name: NA count, dtype: float64
```

This percentage was used to decide if that feature would be deleted or fill the missing values.

- If the percentage for a column was more than 90%, the column was dropped from the data set.

- If the percentage for a column was less than 90%, the missing values in the column were filled depending on the data type of that column.
    - For columns with numerical data type, missing values were filled with -1.
    - For columns with categorical data type, missing values were filled with 'UNK'.

d. **Feature Selection**: [Jay]

At this point in the preprocessing pipeline, the data consisted of 47 columns. Feature selection was performed to reduce the dimensionality of the data. This helped with removing the features that were either irrelevant for prediction or affected the model in some negative manner. Besides helping with the accuracy of the model,this approach significantly decreased the training time for all the models. Otherwise, these times might have been unrealistic and made the process of fine tuning each and every model nearly impossible.

We used Scikit-learn's Extra Trees Classifier to calculate feature importance for every feature. The Extra Trees Classifier calculates the feature importance using a decision criteria, here Information Gain, calculated by each decision tree. The feature that provides the highest information gain is given the highest importance and the total importance sums up to 1. Features like totals.newVisits, trafficSource.isTrueDirect, and visitStartTime were determined to be unimportant by this process and, thus, we removed there features from the data set.

Some of the features had majority/all of the values missing. Most of these columns were detected and removed in the process of Data Cleaning but in case they were missed, their feature importance was calculated as 0 and thus the columns were removed.

Modeling And Evaluation:

a. **Baseline Implementations and Clustering**: [Sanchit]

The modeling process started with baseline implementations. These baseline models were used to examine how the simpler models would perform on the data set before modeling with more advanced techniques. Scikit-learn, a machine learning library for the Python was used for these models.

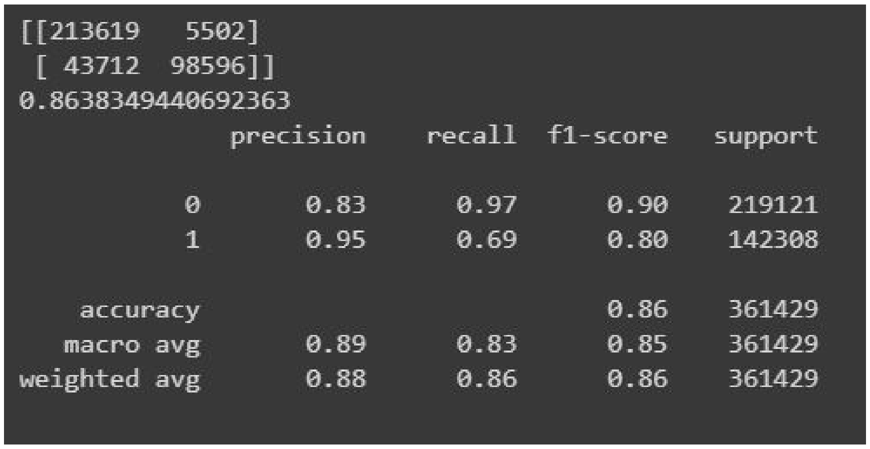
To start the modeling process, first the categorical data set was converted into numerical type using LabelEncoder and LabelBinarizer. From the Feature Selection and

Feature Engineering process in the Data Pre-processing pipeline, we obtained features that would be used for these classification model based on their importance. These features included, 'totals.bounces', 'totals.hits', 'totals.newVisits', 'totals.pageviews' 'totals.sessionQualityDim', 'totals.timeOnSite', 'trafficSource.isTrueDirect', 'visitNumber', 'country_dem', 'latitude' and, 'longitude'. These columns which has numerical data now, were rescaled using StandardScaler or MinMaxScaler depending on the model used.

- **Linear Regression**

  Linear regression is suitable for predicting output that is continuous value, such as predicting the price of a property. Its prediction output can be any real number, range from negative infinity to infinity. The regression line is a straight line. For this data set, we took the default parameters of Linear Regression provided by Scikit-learn. The data set was rescaled using the StandardScalar which normalized the values between -1 to 1. This normalization significantly improved the performance of the model.

  The model was evaluated using metrics provided by Scikit-learn. The model achieved about 87.7% training accuracy and 86.38% validation accuracy. We used confusion matrix along with precision, recall, f1-score and support to evaluate the model which resulted as shown in the figure.



```
[[213619   5502]
 [ 43712  98596]]
0.8638349440692363
              precision    recall  f1-score   support

           0       0.83      0.97      0.90    219121
           1       0.95      0.69      0.80    142308

    accuracy                           0.86    361429
   macro avg       0.89      0.83      0.85    361429
weighted avg       0.88      0.86      0.86    361429
```

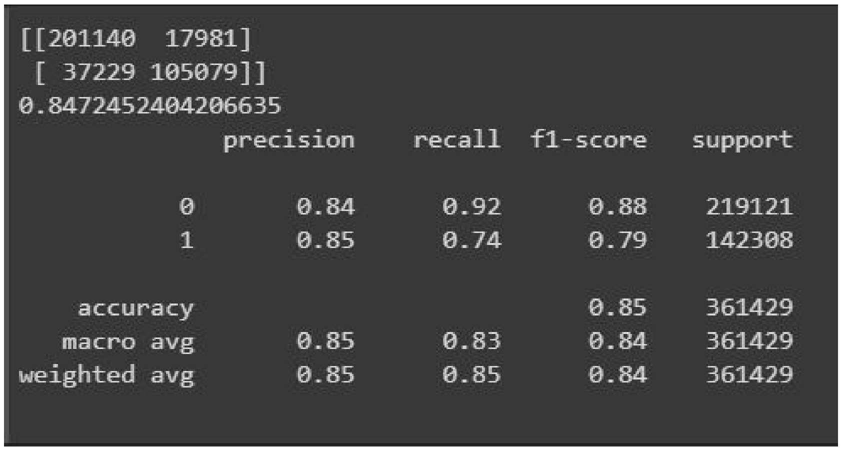
- **Naive Bayes Classifier**

  Naive Bayes Classifier is an algorithm used for binary and multi-class classification problems. We used two variations of Naive Bayes Classifier for the baseline implementations:

  - **Gaussian Naive Bayes Classifier**

Naive Bayes is extended to real-valued attributes, with a Gaussian distribution, to get the first variation of the classifier used in the process. For this data set, we took the default parameters of Gaussian Naive Bayes Classifier provided by Scikit-learn. The data set was rescaled using the StandardScalar which normalized the values between -1 to 1. This normalization improved the performance of the model.

The model was evaluated using metrics provided by Scikit-learn. The model achieved about 88.02% training accuracy and 84.72% validation accuracy. We used confusion matrix along with precision, recall, f1-score and support to evaluate the model which resulted as shown in the figure.

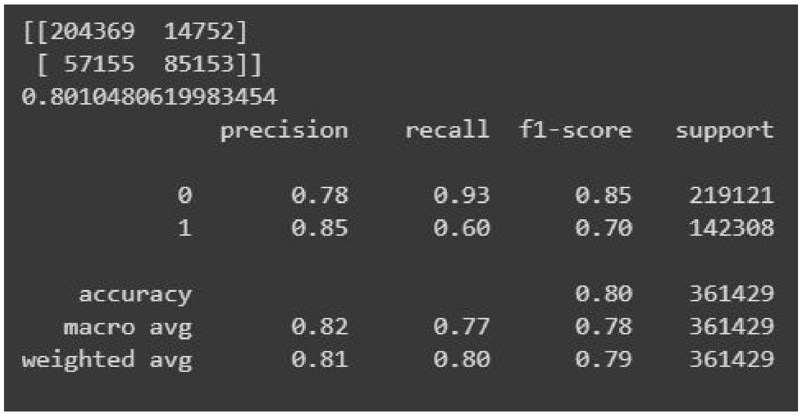


```
[[201140  17981]
 [ 37229 105079]]
0.8472452404206635
              precision    recall  f1-score   support

           0       0.84      0.92      0.88    219121
           1       0.85      0.74      0.79    142308

    accuracy                           0.85    361429
   macro avg       0.85      0.83      0.84    361429
weighted avg       0.85      0.85      0.84    361429
```

- **Multinomial Naive Bayes Classifier**

Naive Bayes with real-valued attributes, uses a Multinomial distribution, which gives us the second variation of the Naive Bayes Classifier used in the process. For this data set, we took the default parameters of Multinomial Naive Bayes Classifier provided by Scikit-learn. The data set was rescaled using the MinMaxScaler which normalized the values between 0 to 1 since Multinomial Naive Bayes Classifier does not allow negative values. This normalization improved the performance of the model.

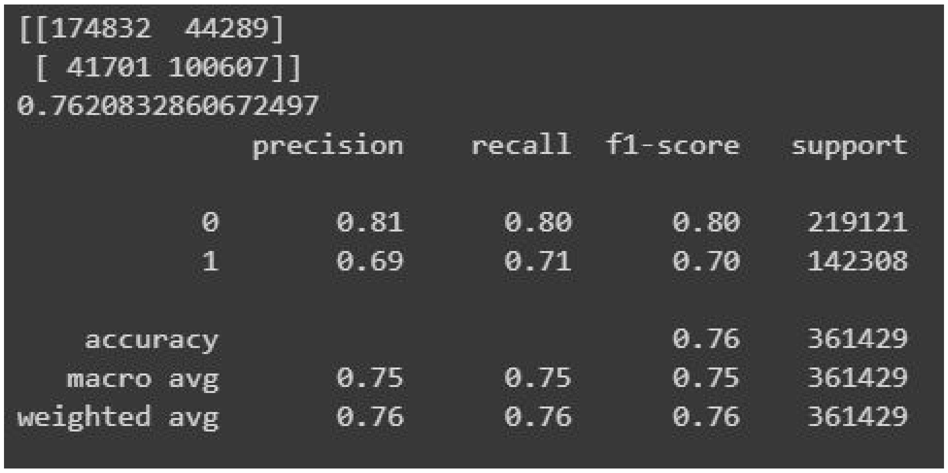
The model was evaluated using metrics provided by Scikit-learn. The model achieved about 83.98% training accuracy and 80.10% validation accuracy. We used confusion matrix along with precision, recall, f1-score and support to evaluate the model which resulted as shown in the figure.


```
[[204369  14752]
 [ 57155  85153]]
0.8010480619983454
              precision    recall  f1-score   support

           0       0.78      0.93      0.85    219121
           1       0.85      0.60      0.70    142308

    accuracy                           0.80    361429
   macro avg       0.82      0.77      0.78    361429
weighted avg       0.81      0.80      0.79    361429
```

- **Clustering**

  We performed Clustering to have an Unsupervised Learning implementation as taught in class. It was a means to explore various models and examine the performance through different types of models. For Classification using clustering, we used KMeans Clustering provided by Scikit-Learn. We formed two clusters for each class label, with all the default parameters. The data set was rescaled using the MinMaxScaler which normalized the values between 0 to 1 for KMeans.

  The model was evaluated using metrics provided by Scikit-learn. The model achieved about 79.84% training accuracy and 76.2% validation accuracy. We used confusion matrix along with precision, recall, f1-score and support to evaluate the model which resulted as shown in the figure.


```
[[174832  44289]
 [ 41701 100607]]
0.7620832860672497
              precision    recall  f1-score   support

           0       0.81      0.80      0.80    219121
           1       0.69      0.71      0.70    142308

    accuracy                           0.76    361429
   macro avg       0.75      0.75      0.75    361429
weighted avg       0.76      0.76      0.76    361429
```

b. **SVM**: [Jay]

Another model that we used to predict customer retention was Linear Support Vector Machine/Classifier. We used the scikit-learn library for this task. The configuration that was used for training the model is penalization using ‘l2 norm’, loss function used ‘squared hinge’, criteria for convergence was 10^{-4}, regularization parameter was set to 1.0 and maximum iterations was set to 1000. Other values of the regularization parameter that were tested were 0.1 and 100 but were found to provide similar results. So, we used the default in the end. The problem is a two class problem and therefore the multi_class strategy of ‘ovr’(One vs Rest) did not matter much for our problem.

The following features were selected based on the feature importances obtained as part of the Feature Selection and Feature Engineering Process. 'totals.bounces', 'totals.hits', 'totals.newVisits', 'totals.pageviews' 'totals.sessionQualityDim', 'totals.timeOnSite', 'trafficSource.isTrueDirect', 'visitNumber', 'country_dem', 'latitude' and, 'longitude'. As can be seen the features that were generated as part of feature engineering are used besides the features that garnered the highest importance. The model assigned the following weights to the features as a result of the training:

● totals.bounces:	-0.0188
● totals.hits:	0.0055
● totals.newVisits:	-0.675
● totals.pageviews:	0.0457
● totals.sessionQualityDim:	0.0088
● totals.timeOnSite:	0.0683
● trafficSource.isTrueDirect:	0.0428
● visitNumber:	0.0472
● country_dem:	0.0385
● latitude:	0.0076
● longitude:	-0.0086

Model Evaluation: Below are the evaluation results for the testing data.

```
Confusion Matrix:    array([[213398,   5723],
                            [ 43515,  98793]])
```

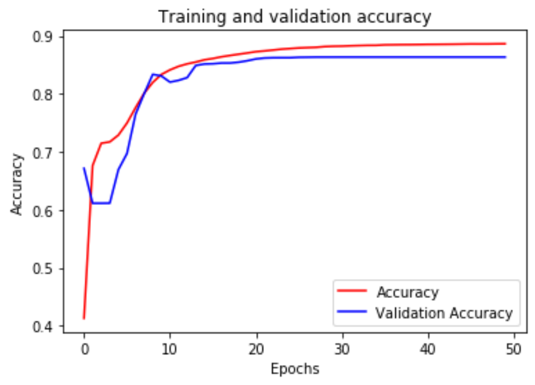
```
Classification Report:
                  precision    recall    f1-score    support

              0      0.83        0.97       0.90      219121
              1      0.95        0.69       0.80      142308
```

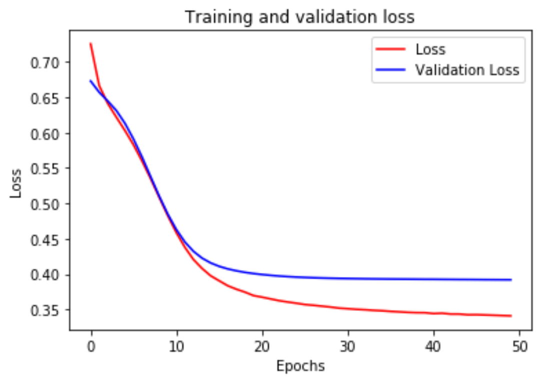
```
    accuracy                           0.86    361429
   macro avg       0.89      0.83      0.85    361429
weighted avg       0.88      0.86      0.86    361429
```

c. **Neural Networks**: [James]

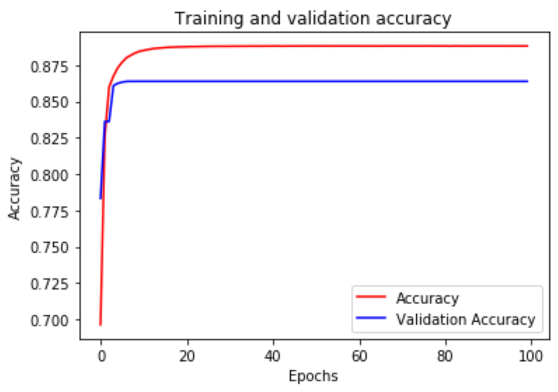
To develop, train, and test neural network models we made use of TensorFlow with Keras. The first step was constructing various models that could then be used for customer retention prediction. We tested models with zero, one, and two hidden layers and found the best models were those with one or two hidden layers with the difference in accuracy being negligible. The activation function used at the input and hidden layers was chosen to be ReLU (Rectified Linear Unit) which is considered to be one of the better choices for an activation function as it does not suffer from the phenomenon of vanishing gradients (as has been observed with the activation function tanh). Since the neural network is performing binary classification the output layer is a single node with a sigmoid activation function. This way, the final layer will output values in the range [0,1] where values in the range [0,0.5) indicate the model has predicted the customer will not return and values in the range [0.5,1] indicate the model has predicted that the customer will return.

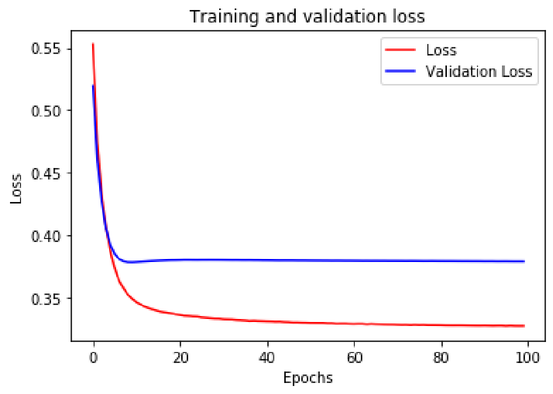
A common problem observed with neural networks is overfitting of the training data which results in poor performance on new data samples. To avoid overfitting in our model, we added dropout layers after each activation function. These layers drop percentage of neurons, specified by the user, at each iteration of the training phase resulting in the value of certain weights in the model not contributing to the output value and, accordingly, not being updated at each iteration in which they are dropped. As such, these dropout layers help prevent the model from becoming too dependent on a few of the learned weights in the model. After adding the dropout layers we observed an improved model that did not suffer from overfitting of the training data. The models were training on Google Colab where we were able to make use of a GPU for accelerating the training process in TensorFlow. The models were first trained on a subset of the data to check that decent performance could be obtained before scaling up and training on the entire training set. Training and testing accuracy and loss plots were generated during the process, provided below, and indicated that the proposed model was not overfitting the data and was performing well on both the training and testing set.



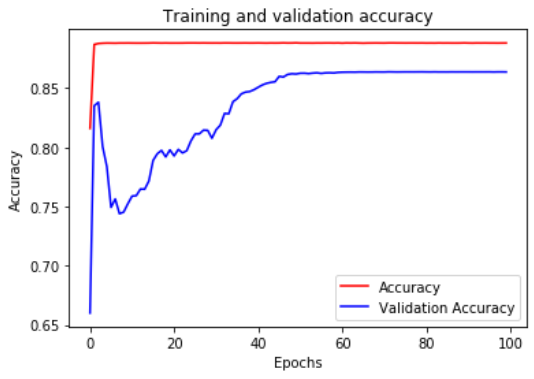


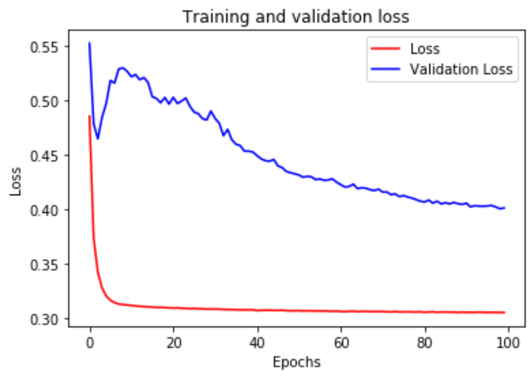
The mode was then trained on the full data set and achieved an accuracy of 88.6% on the training set and an accuracy of 86.3% on the testing set. Plots for the training and validation accuracy and loss on the full sets are provided below.





Following this, we experimented with adding an additional component to our neural network called batch normalization. The idea of batch normalization is to normalize the data before applying each activation function so that the average of each component is zero. Following this normalization, when the ReLU activation function is applied the values below zero are dropped which results in less information for the neural network to process. This idea was introduced in the literature a few years ago and has lately become a somewhat standard practice. The main advantage to batch normalization is that the model will be able to train faster as the normalization typically results in less computation at each layer of the neural network. Occasionally batch normalization will also result in improved accuracy of the model, however, we only observed a reduced training time for each epoch of approximately 20 seconds down from 40 seconds. Plots displaying training and testing accuracy and loss over epochs for the batch normalized model are provided below.



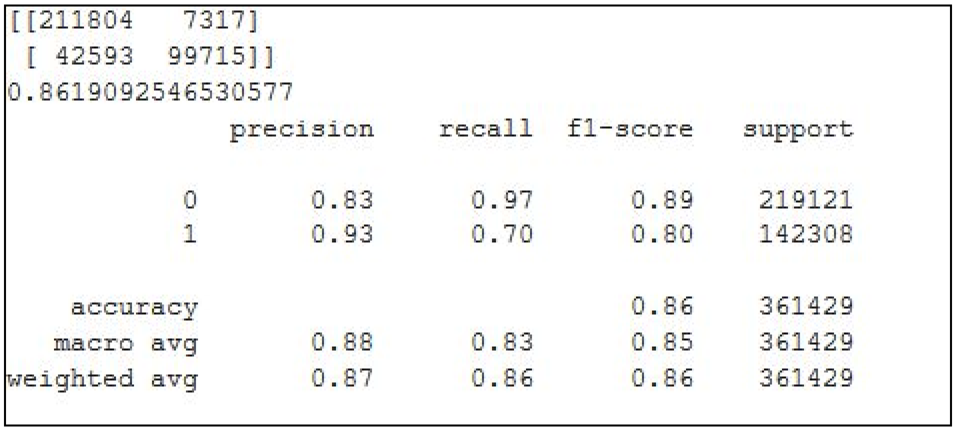


d. **Trees**: [Anshika]

Random Forests:

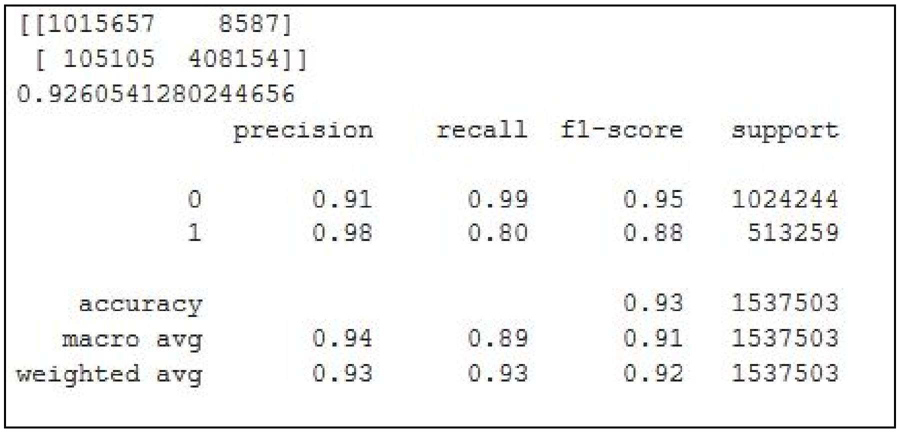
Random forests are ensembles of decision trees trained by bagging (boot-strap aggregation). In this procedure a given number n of bootstraps are generated and each tree learns on independent set of bootstraps.

To begin with, a simple Random Forest was trained with mostly default parameters and 100 estimation trees in the ensemble. The model gave the accuracy of 86.10% and the results were (confusion matrix, accuracy, precision, recall and F1 score) :


```
[[211804   7317]
 [ 42593  99715]]
0.8619092546530577
              precision    recall  f1-score   support

           0       0.83      0.97      0.89    219121
           1       0.93      0.70      0.80    142308

    accuracy                           0.86    361429
   macro avg       0.88      0.83      0.85    361429
weighted avg       0.87      0.86      0.86    361429
```

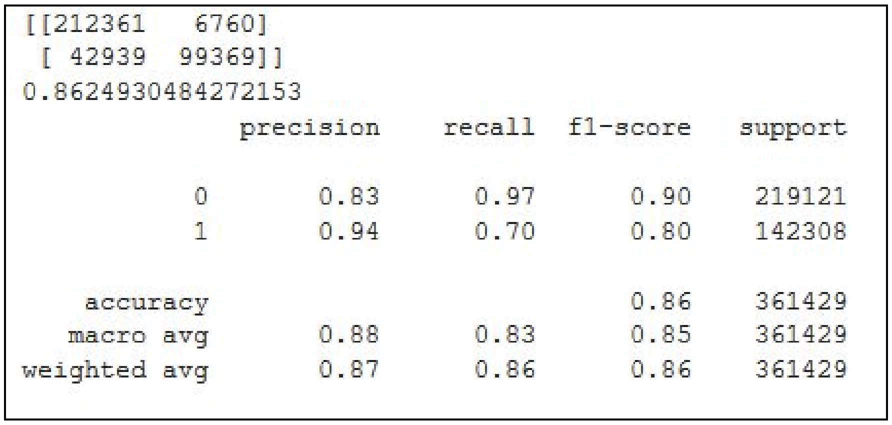
After hypertuning the parameters, the optimum accuracy of 86.24% was reached with 300 estimation trees in the ensemble. Any further hypertuning did not create any significant improvement in the accuracy of the model.

The Training confusion matrix, accuracy, precision, recall and F1 score of the model is :



```
[[1015657    8587]
 [ 105105  408154]]
0.9260541280244656
              precision    recall  f1-score   support

           0       0.91      0.99      0.95   1024244
           1       0.98      0.80      0.88    513259

    accuracy                           0.93   1537503
   macro avg       0.94      0.89      0.91   1537503
weighted avg       0.93      0.93      0.92   1537503
```

The Testing confusion matrix, accuracy, precision, recall and F1 score of the model is :



```
[[212361   6760]
 [ 42939  99369]]
0.8624930484272153
              precision    recall  f1-score   support

           0       0.83      0.97      0.90    219121
           1       0.94      0.70      0.80    142308

    accuracy                           0.86    361429
   macro avg       0.88      0.83      0.85    361429
weighted avg       0.87      0.86      0.86    361429
```

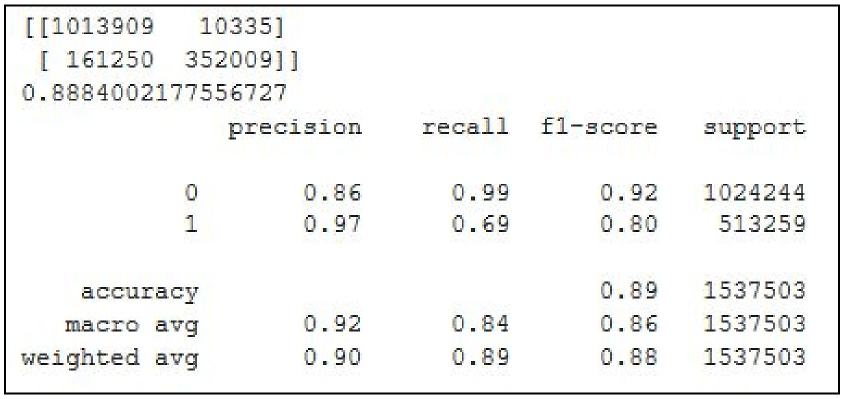
The training accuracy of 92% and testing accuracy of 86% shows that the model is not overfitted.

XGBoost Trees:

After seeing the good performance of trees with the highly imbalanced fraud data, XGB trees became a natural choice to experiment with. XGBoost stands for “Extreme Gradient Boosting” and is decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting frame-work. They are very fast in implementation.

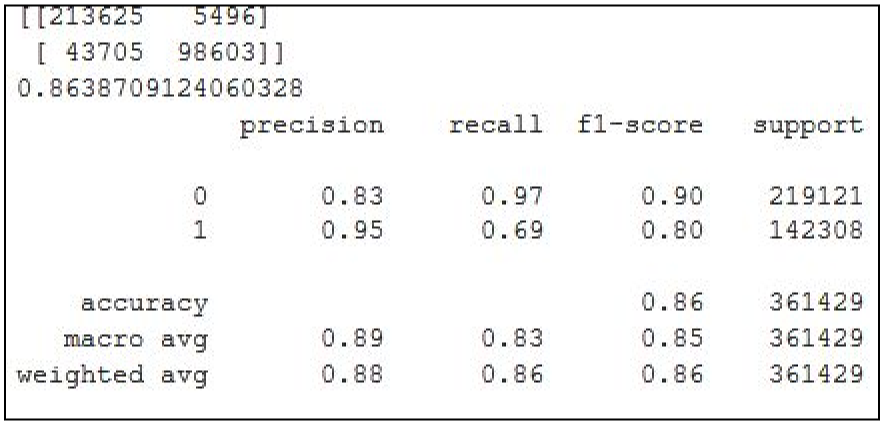
The XGBoost trees provided similar results as Random Forest, but much faster in training and testing. The model was also more compact and occupied lesser memory than Random Forests.

The Training confusion matrix, accuracy, precision, recall and F1 score of the model is :



```
[[1013909   10335]
 [ 161250  352009]]
0.8884002177556727
              precision    recall  f1-score   support

           0       0.86      0.99      0.92   1024244
           1       0.97      0.69      0.80    513259

    accuracy                           0.89   1537503
   macro avg       0.92      0.84      0.86   1537503
weighted avg       0.90      0.89      0.88   1537503
```

The Testing confusion matrix, accuracy, precision, recall and F1 score of the model is :

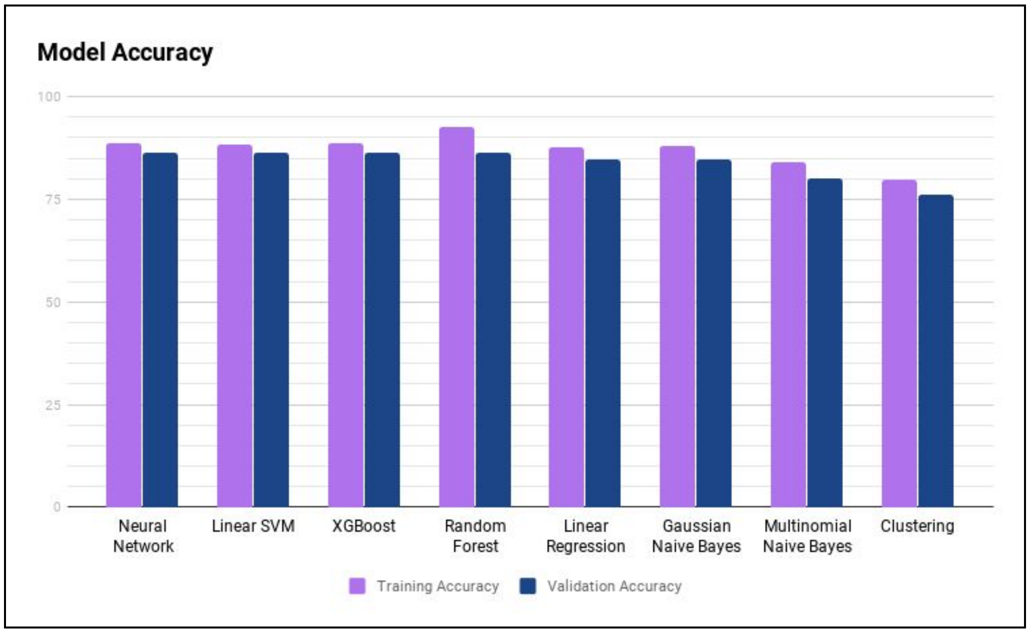


```
[[213625   5496]
 [ 43705  98603]]
0.8638709124060328
              precision    recall  f1-score   support

           0       0.83      0.97      0.90    219121
           1       0.95      0.69      0.80    142308

    accuracy                           0.86    361429
   macro avg       0.89      0.83      0.85    361429
weighted avg       0.88      0.86      0.86    361429
```

The training accuracy of 88% and testing accuracy of 86% shows that the model is not overfitted.

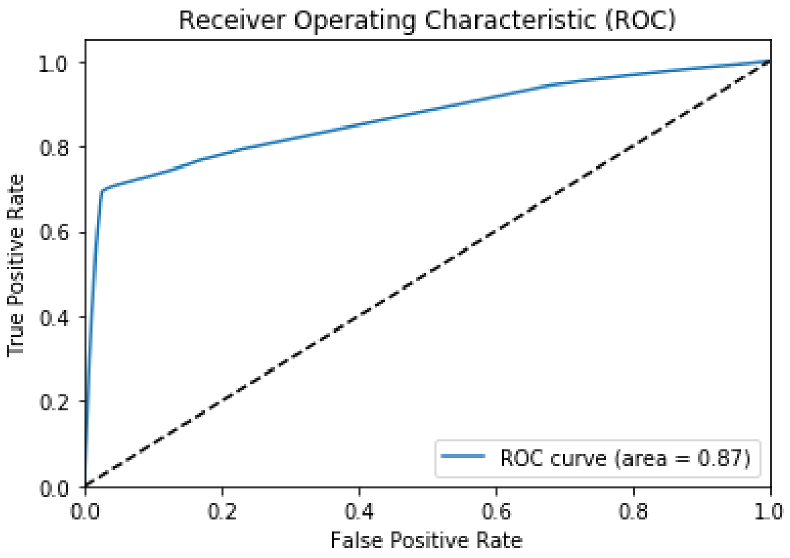
e. **Comparison of Models**: [James]

Following the completion of all of the models, we plotted a bar graph that made it easy to compare how the models all performed on the data and to visualize the relative performance of the models against each other.



As observed by the bar graph, the best performance was achieved by the advanced modeling methods: Neural Networks and Linear Support Vector Machines (SVM). These models achieved an accuracy of nearly 87% on the validation data. Additionally, this bar graph allows us to see that none of the models are really suffering from overfitting of the training data which means that we expect them to perform similarly on future unseen data.

For the best performing model, the neural network, we created a Receiver Operating Characteristic (ROC) curve to illustrate the model's effectiveness on the validation data set.



Post Presentation Improvements:

a. **Ensemble Methods**

Ensemble_1: Created an ensemble of all the models outlined in the Trees section, Random Forest and XGBoosted Trees, by using the Voting Classifier. The classifier predicts the output class label based on the argmax of the sums of the predicted probabilities.

Both models have similar accuracy rates as an individual model i.e. ~86.5%. As an Ensemble model their accuracy remain near the same number (~86.4%) but now we have more reliability as both models have different methods to reach to the final label so there is less chances of the model being biased.

Ensemble_2: An ensemble Voting Classifier using the Support Vector Machine, Logistic Regression, Random Forest and Gaussian Naive Bayes classifiers was created for prediction and then hard voting. The training accuracy and testing accuracy for the Classifier ended up being 88.88 and 86.37, respectively, which performs as well as the best neural network model.

Member Contributions:

Each person in this group contributed an equal amount of work to each component of the project. The data preprocessing was split into four equal parts and each person in the group was responsible for one part of the data preprocessing. The modeling and evaluation was also divided into four equal parts and each person was responsible for completing one part of the modeling/evaluation phase of the project. Finally, each person was responsible for making presentation slides and writing the parts of the report that pertained to the work they completed on the data preprocessing and modeling/evaluation. Everyone worked together well and completed their work in a timely manner.