Final Report (Amex Dataset)

Imputing missing values:

- The data set had a lot of missing values and they had different kind of labels like ‘missing’, ‘na’ and ‘N/A’. So, the first step was to replace all of these with np.NaN.
- An R script was used to find the percentage of missing values in every column. We separately found out the percentage of missing values of the columns for the positive points and for the negative points. Those columns having more than 40 percent missing values in both the categories were eliminated.

R Code for the above procedure:

```
data=read.csv("T1.csv")
i=1, j=1
 X=rep(0,times=49),  Y=rep(0,times=49)
 count1 = 0, count2 = 0
 while(j<=49){
   count1 = count2 = 0
   i=1
  while(i<=83000){
    if(is.na(data[i,j])){
      if(data[i,50]==0){
         count1=count1+1
      }else{
        count2=count2+1
      }
    }
    i=i+1
  }
   X[j]=count1
   Y[j]=count2
   j=j+1
 }

 i=1
 while(i<=83000){
   if(data[i,50]==0){
     count=count+1
    }else{
         c=c+1
    }
```

```
  i=i+1
 }

 j=1
 while(j<=49){
  X[j]=X[j]/59145*100
  Y[j]=Y[j]/23855*100
  j=j+1
 }
```

- Then the same columns were eliminated from the test data set. Then both the data sets were combined so that we get a better idea of the correlation that exists between the remaining columns.
- There was a categorical variable which contained only two values ‘L’ and ‘C’. They were replaced as 0 and 1 respectively.
- The random forest imputer was used to impute the remaining missing values.

There are three general strategies that are most commonly used for RF missing data imputation:

A. Preimpute the data; grow the forest; update the original missing values using proximity of the data. Iterate for improved results.

B. Simultaneously impute data while growing the forest; iterate for improved results.

C. Preimpute the data; grow a forest using in turn each variable that has missing values; predict the missing values using the grown forest. Iterate for improved results.

We have used the miss forest imputer which uses strategy C. Strategy C does not use outcome data at all and is solely a technique for imputing features (**X** variables), while strategies A and B utilize outcome information in their model building. Since we have combined both the test and training data, we do not have outcome values for all the data. So, miss forest was used for our case.

The missForest algorithm imputes the missing data problem as a prediction problem. Data is imputed by regressing each variable in turn against all other variables and then predicting missing data for the dependent variable using the fitted forest. With *p* variables, this means that *p* forests must be fit for each iteration, which can be slow in certain problems.

- Once the imputations are done, the data sets are separated into the labelled and non-labelled data set.
- The labelled dataset is then split into 80% train and 20% test data and the features are scalled.

Balancing the Dataset

- The given dataset is skewed. There are a lot of occurrences of the class '0' in the predictions. About 70% of the predictions are zeroes.
- Due to the skewed nature of the dataset if we train a classifier on this dataset, it will be biased towards predicting the most frequent class.
- In order to balance the dataset, we need to do oversampling of the minority class. We do this using SMOTE (Synthetic Minority Oversampling Technique).
- SMOTE synthesizes new minority class instances between existing (real) minority instances. It may be visualized as SMOTE draws lines between the existing minority classes and then it synthesizes new minority class instances somewhere on these lines.

Classifier

- As a first step the LogisticRegression and the NaiveBayes classifiers were trained on the dataset. Using this we had obtained a balanced score of 0.63. This model provided a good baseline for comparing the scores with the more sophisticated models.
- Then we used the ensemble models such as XGBoost, RandomForest and AdaBoost. The balanced scores of these models were also documented.
- In order to get a better score, we used an ensemble vote classifier. This classifier is available in the sklearn library.
- The EnsembleVoteClassifier is a meta-classifier for combining similar or conceptually different machine learning classifiers for classification via majority voting.
- In this we have implemented the 'hard' voting. In hard voting, we predict the final class label as the class label that has been predicted most frequently by classification models.
- In our classifier we have used a combination of 5 XGBoost, 2 RandomForest, 1 AdaBoost and 1 LogisticRegression classifiers in the ensemble vote classifier.
- We had manually increased the number of classifiers to determine the combination that was giving the best results.

Test Metrics:

Confusion matrix	[[8676 3080] [1591 3253]]
Accuracy	0.7186144578313253
Balanced accuracy	0.7047792802677284
F1 score :	0.5820882168739375