Project Report

Introduction to Data Analytics - MS4610

Group 25 :

1. Vyshnav P - CE18B061
2. Hari kishore P - NA18B019
3. Souridas A - AE18B046
4. Jishnu N - NA18B110
5. Thandava Krishna - NA18B117
6. Murali Krishna - NA18B018

1. Problem Statement :

The problem statement is to develop a credit risk model that predicts the likelihood of a customer to default a payment. We are given customer data (feature vector of size 47) and we need to predict the likelihood of the customer defaulting after 12 months. Thus we could sum up that the project is a binary classification problem.

2. Data interpretation & Visualization

The dataset consists of 83000 data points each with 47 features. They are labeled from ‘mvar1’ till ‘mvar47’. ‘mvar47’ is categorical with categories ‘C’ or ‘L’.

amex_test_df

	application_key	mvar1	mvar2	mvar3	mvar4	mvar5	mvar6	mvar7	mvar8	mvar9	mvar10	mvar11	mvar12	mvar13	mvar14
0	578069	1719	0.6174	8.623	0.000	0.000	258	258	258	10729	307	1141	40	0	49550
1	578070	1795	0.2051	0.000	0.000	0.000	1685	12711	8913	80519	18099	missing	3457	455	198200
2	578071	1742	0.5082	0.000	0.000	0.000	1185	8954	8954	1189	1185	missing	3028	1453	122884
3	578072	1685	0.2595	25.409	0.000	0.000	missing	3354	missing	missing	3354	4231	missing	missing	118920
4	578073	1666	1.2678	0.000	0.000	0.000	570	570	570	missing	570	missing	74	missing	42613
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
46995	310027	1736	2.1740	0.000	0.000	0.000	11	4248	1577	13379	6671	missing	11791	9768	247750
46996	310028	1724	0.0000	1.108	0.768	0.000	missing	64041	missing	10926	84839	3329	2310	3222	77298
46997	310029	1605	0.2901	11.561	0.937	2.976	missing	2277	missing	3964	5709	missing	269	2619	11892
46998	310030	1780	1.1874	0.000	0.000	0.000	0	6356	4802	3206	18180	24	2665	12343	84235
46999	310031	1727	1.9288	1.441	0.000	0.000	0	25773	2869	132985	71788	missing	20839	15084	83244

Fig 1. Glimpse of the first 14 variables of the dataset

Train data dimensions are : 83000 x 49

```
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   application_key  83000 non-null  int64
 1   mvar1            83000 non-null  object
 2   mvar2            77114 non-null  float64
 3   mvar3            82465 non-null  float64
 4   mvar4            82465 non-null  float64
 5   mvar5            82465 non-null  float64
 6   mvar6            83000 non-null  object
 7   mvar7            83000 non-null  object
 8   mvar8            83000 non-null  object
 9   mvar9            83000 non-null  object
 10  mvar10           83000 non-null  object
 11  mvar11           83000 non-null  object
 12  mvar12           83000 non-null  object
 13  mvar13           83000 non-null  object
 14  mvar14           83000 non-null  int64
 15  mvar15           83000 non-null  object
 16  mvar16           83000 non-null  object
 17  mvar17           83000 non-null  object
 18  mvar18           83000 non-null  object
 19  mvar19           83000 non-null  object
 20  mvar20           83000 non-null  object
 21  mvar21           59538 non-null  float64
 22  mvar22           52332 non-null  float64
 23  mvar23           40689 non-null  float64
 24  mvar24           63470 non-null  float64
 25  mvar25           83000 non-null  object
 26  mvar26           83000 non-null  object
 27  mvar27           83000 non-null  object
 28  mvar28           83000 non-null  object
 29  mvar29           83000 non-null  object
 30  mvar30           83000 non-null  object
 31  mvar31           83000 non-null  object
 32  mvar32           83000 non-null  object
 33  mvar33           81131 non-null  float64
 34  mvar34           83000 non-null  object
 35  mvar35           83000 non-null  object
 36  mvar36           83000 non-null  object
 37  mvar37           83000 non-null  object
 38  mvar38           83000 non-null  object
 39  mvar39           83000 non-null  object
 40  mvar40           83000 non-null  object
 41  mvar41           83000 non-null  object
 42  mvar42           83000 non-null  object
 43  mvar43           83000 non-null  object
 44  mvar44           74851 non-null  float64
 45  mvar45           83000 non-null  object
 46  mvar46           83000 non-null  object
 47  mvar47           83000 non-null  object
 48  default_ind      83000 non-null  int64
dtypes: float64(10), int64(3), object(36)
```

There are quite a few missing values in most of the columns. Thorough data cleaning is required to train the model on this dataset.
First we should convert the non numeric values to NaN values. Noise in the data is in the form of strings like ‘na’ or ‘missing’. Such object data types have to be converted to float or int data types. For this we either remove them altogether or convert them to NaN values and then fill it with the mean of the column or mode value depending upon if it is a continuous variable or a categorical one.
It will also be important to remove outliers as it will delay the convergence of the model. We should also standardise the continuous variables so that model convergence is faster.

3. Data Cleaning and feature selection

The dataset has to be properly cleaned in order to train a robust model. The data had a lot of missing values in a lot of columns.

First we converted the cells with ‘missing’ , ‘na’ or any other non numerical values to NaN or np.nan values from the continuous variables.

```
for col in category_columns[:-1]:
    values = amex_df[col]
    k = 0
    L = set()

    for i in range(len(values)):
        if (not is_number(values[i])):
            L.add(values[i])
    L = list(L)
    amex_df[col] = amex_df[col].replace({L[0]:np.NaN})
    amex_test_df[col] = amex_test_df[col].replace({L[0]:np.NaN})
```

The last variable ‘mvar47’ is a categorical variable with two classes C or L, so we will save it for the end to convert it to one hot encodings

```
amex_df[category_columns[:-1]] = amex_df[category_columns[:-1]].apply(pd.to_numeric)
amex_test_df[category_columns[:-1]] = amex_test_df[category_columns[:-1]].apply(pd.to_numeric)
```

Removal of NaN values

The percentage of NaN values in each column was found out and only those features with less than 7% NaN were selected to the train set.

```
[ ]  input_cols = []
     index = amex_df.isnull().sum().index
     j =0 ;
     for i  in  (amex_df.isnull().sum()/len(amex_df)*100 <7):
         if i:
             input_cols.append(index[j])
         j+=1
```

Outlier removal based on z score

Once this was done we calculated z score using stats.zscore() from the scipy library.
The standard value of z_score = 3 was used to identify the outliers. Any datapoint with z score more than 3 is considered to be an outlier.

Removing outliers based on z-score

```
from scipy import stats
print(temp_df.shape)
temp_df = temp_df[(np.abs(stats.zscore(temp_df[input_cols[:-1]])) < 3).all(axis=1)]
print(temp_df.shape)
```

```
(63137, 26)
(50474, 26)
```

Train test split

After all the data cleaning we get data points with feature vectors of size 26. This set is split into train and

Train test split

```
train_df, val_df = train_test_split(temp_df, test_size=0.25, random_state=42)
test_df = temp_test_df
```

```
print('train_df.shape :', train_df.shape)
print('val_df.shape :', val_df.shape)
print('test_df.shape :', test_df.shape)
```

```
train_df.shape : (37855, 26)
val_df.shape : (12619, 26)
test_df.shape : (47000, 25)
```

validation sets with a split ratio of 0.25.
The test set for leaderboard submission also undergoes the data imputation like the train set.
The final shapes of all the three data sets are printed.

Standardizing features

We convert the continuous variable to standard normal distribution by removing the mean and scaling the data points to unit variance for each column.

$$z = \frac{(x-\mu)}{\boldsymbol{\sigma}}$$

$$\mu = \frac{1}{N}\boldsymbol{\Sigma}(x) \qquad \boldsymbol{\sigma} = \sqrt{\frac{1}{N}\boldsymbol{\Sigma}(x - \mu)^2}$$

Here μ is the mean of the training sample or zero and $\boldsymbol{\sigma}$ is the standard deviation of the training samples.

Standardizing features

```
[ ]  from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()
     scaler.fit(temp_df[numeric_cols])
     train_inputs[numeric_cols] = scaler.transform(train_inputs[numeric_cols])
     val_inputs[numeric_cols] = scaler.transform(val_inputs[numeric_cols])
     test_inputs[numeric_cols] = scaler.transform(test_inputs[numeric_cols])
```

One hot encoding

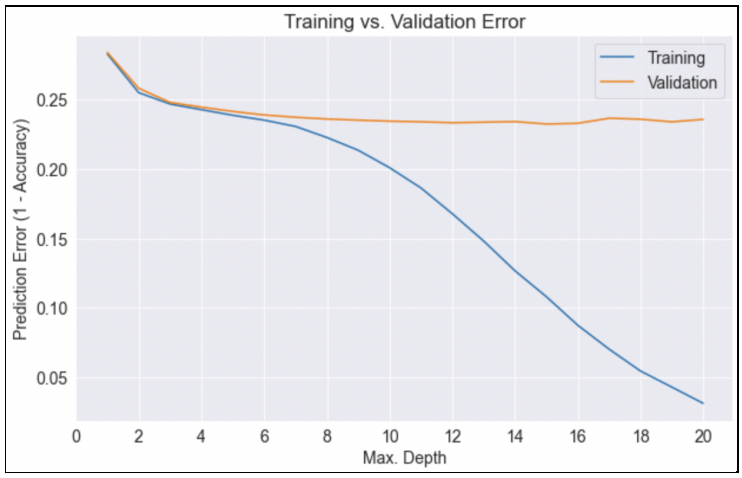
The categorical variables are converted to one hot encodings using the method from the sklearn.preprocessing library.

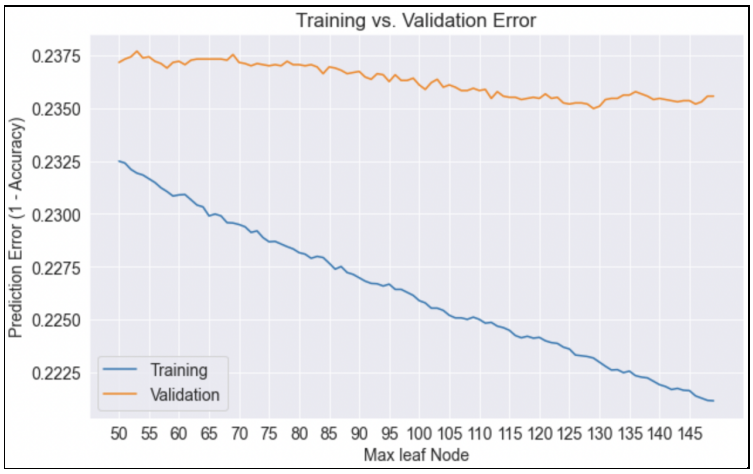
```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(sparse = False,handle_unknown='ignore')
encoder.fit(temp_df[categorical_cols])
encoded_cols = list(encoder.get_feature_names_out(categorical_cols))
train_inputs[encoded_cols] = encoder.transform(train_inputs[categorical_cols])
val_inputs[encoded_cols] = encoder.transform(val_inputs[categorical_cols])
test_inputs[encoded_cols] = encoder.transform(test_inputs[categorical_cols])
```

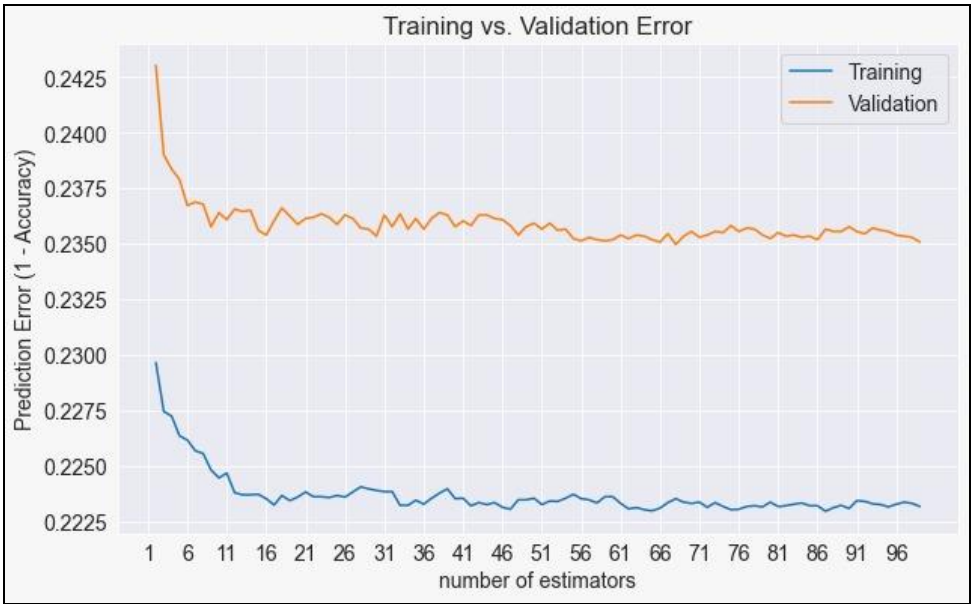
5. Training and model selection

5.1 Random Forest

Hyperparameters of random forest classifier are :
max_depth : The maximum depth of the tree.
max_leaf_nodes : The trees are grown with that many leaf nodes.
N_estimators : The number of trees in the forest.







We have used **learning curves** to find the optimal parameters for the model.

After going through the plots we have arrived at the best parameters as

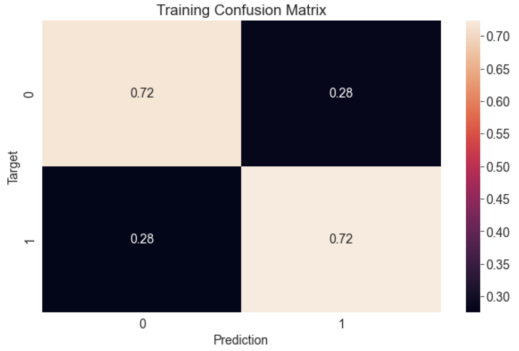
best_params : {‘max_depth’ : 12, ‘max_leaf_nodes’: 128, ‘n_estimators’:68}

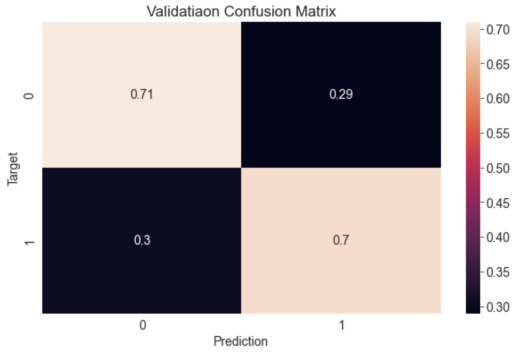
After using ‘gini’ impurity to measure the quality of a split we train the model to arrive at the following performance metrics on the validation set.

As we can see the model performs quite well on the train and validation sets. The type I & II errors are low as well.

We will compare the performance with other models before building the final model.

```
test_param(**best_params,class_weight = {1:2.53,0:1})

Training Accuracy: 71.84%
Validatiaon Accuracy: 70.62%
(71.8389889012798, 70.61915887850468)
```



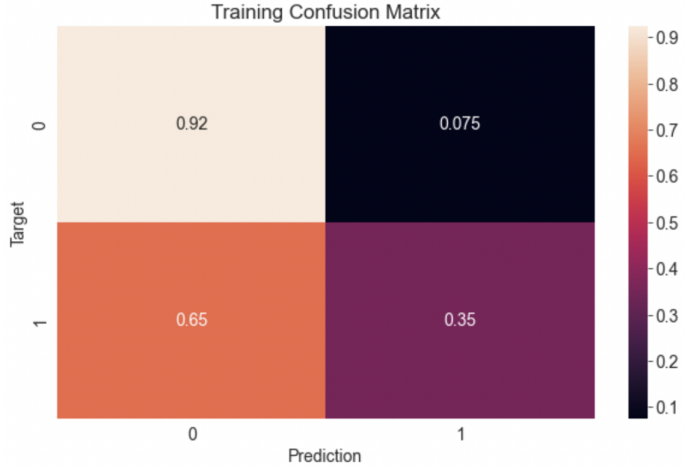


5.2 Logistic regression

We had trained a Logistic regression model on the preprocessed dataset. We had used ‘saga’ as the solver since it is good for large datasets.

As we can see from the confusion matrix even though the precision of the model is good the Type 1 error (0.65) associated with the predictions are quite high.

```
test_param(solver ='saga',max_iter = 1000)

Training Accuracy: 76.22%
Validatiaon Accuracy: 76.49%
(76.2182925318181, 76.48683092608326)
```

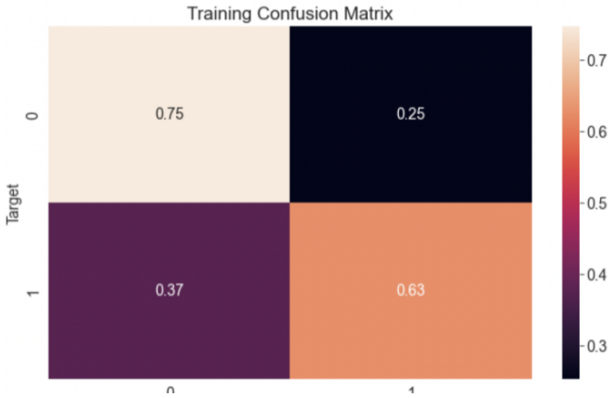


5.3 Gaussian Naive bayes classifier

Using the GaussianNB library from sklearn.naive_bayes package a basic Gaussian Naive Bayes model was trained.

```
def test_param(**param):
    m = GaussianNB(**param)
    m.fit(X_train, train_targets)
    k = predict_and_plot(m,X_train, train_targets, 'Training')
    v = predict_and_plot(m,X_val, val_targets, 'Validatiaon')
    return k, v

val = test_param()
```

```
Training Accuracy: 71.28%
Validatiaon Accuracy: 71.26%
```

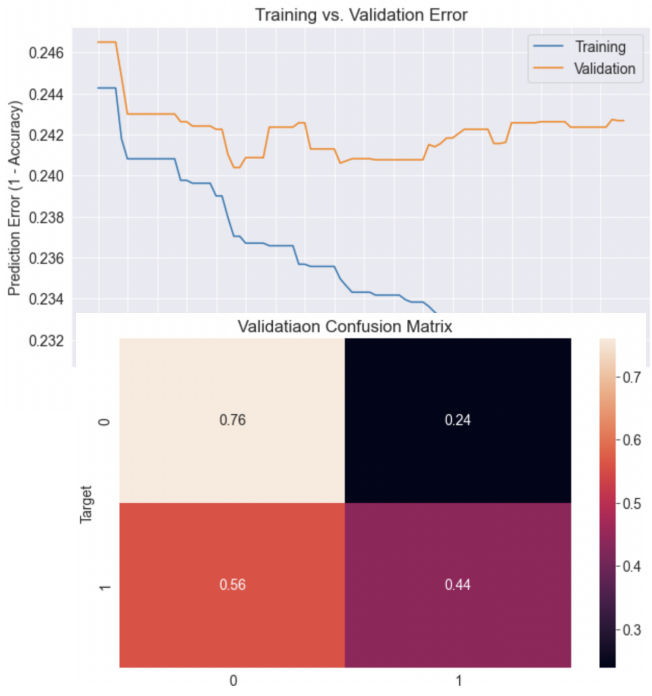


5.4 Decision tree model

Since our dataset is quite complex a decision tree model would perform better.
Hence we had spent quite some time in training and fine tuning a decision tree on the dataset.

The model parameters were chosen on the basis of learning curves.
For finding the optimal depth of the decision tree the training and validation errors for decision trees of depth 0 to 20 were plotted.
Similarly the max_leaf_nodes parameter was changed from 10 to 95 to obtain the optimal value of 34.

However it slightly falls short in performance on the validation set compared to random forest model.
The model exhibited high variance and performed poorly on the leaderboard.



6. Final Model

Dealing with missing data was an important task in the project to increase the final model accuracy. We had build two models, one that works only on clean data, other one that works on data with missing values filled with the column mean.

Each models was separately tuned and hence had different final parameters.

Model 1 :

```
[ ]  model = RandomForestClassifier(random_state=42,n_estimators= 68,max_depth=12,max_leaf_nodes=128,class_weight = {1:ratio_1,0:1})
```

Model 2 :

```
model = RandomForestClassifier(random_state=42,n_estimators = 83,max_depth=8,max_leaf_nodes=130,class_weight = {1:2.47,0:1})
```

Thus to build two models we had made two copies of the dataset one with all the NaN rows dropped and the other underwent data imputation. Here temp_df and temp_test_df are the ones that are used to train the first model, while temp_miss_df and temp_test_miss_df further undergo data imputation before they are used to training the second model.

```
creating 2 data set . one containing the missing values and other with out any missing values

[ ]  temp_df = amex_df[input_cols].copy()
     temp_test_df = amex_test_df[input_cols[:-1]].copy()
     temp_test_miss_df = amex_test_df[input_cols[:-1]].copy()
     temp_miss_df = amex_df[input_cols].copy()
```

6.1 Data imputation

We had performed the basic imputation of filling the missing values with the mean of the corresponding columns.

	col1	col2	col3	col4	col5
0	2	5.0	3.0	6	NaN
1	9	NaN	9.0	0	7.0
2	19	17.0	NaN	9	NaN

mean()

	col1	col2	col3	col4	col5
0	2.0	5.0	3.0	6.0	7.0
1	9.0	11.0	9.0	0.0	7.0
2	19.0	17.0	6.0	9.0	7.0

```
imputer.fit(temp_miss_df[numeric_cols])
```

```
from sklearn.impute import SimpleImputer
```

```
imputer = SimpleImputer(strategy = 'mean')
```

6.2 K fold cross validation

Cross validation was performed to double check the tuned model

```
from sklearn.model_selection import KFold

kfold=KFold(n_splits=5, shuffle=True, random_state=43)

scores = cross_val_score(model, X, Y, cv=kfold)
```

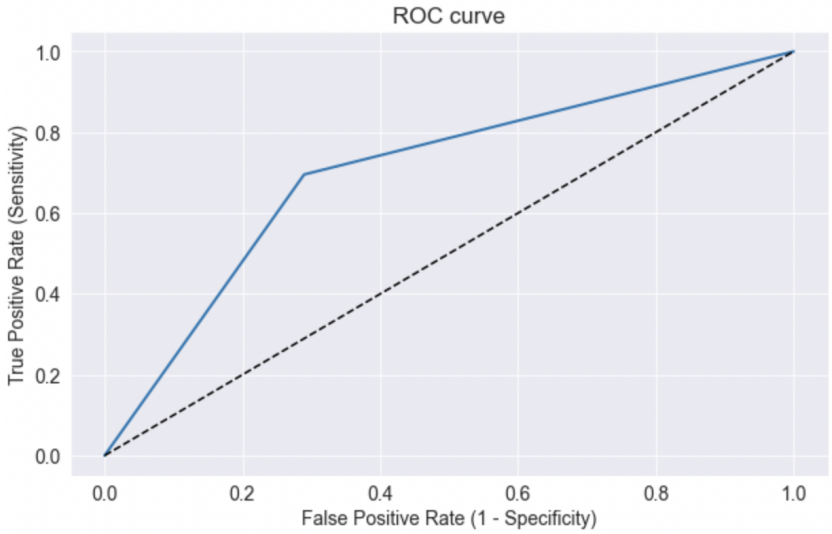
```
scores
```

```
array([0.70116163, 0.70394955, 0.69605045, 0.70029871, 0.6990375 ])
```

parameters for high variance. The K fold cross validation gave us consistent scores close to 0.7 thus indicating that the model is robust enough for new data.

6.3 ROC curve

As we can see the model has an AUC close to 0.7, thus it can distinguish between 0 and 1 classes 70% of the time. Since this is an acceptable auc score for a binary classification we can consider this model for final predictions.



6.4 Class Weights

Since there is class imbalance in the dataset we had used class weights while training the model. The ratio between the number of classes was used for the same.

```
ratio_1 = Y.value_counts()[0]/Y.value_counts()[1]
ratio_1
```

```
2.535389092274477
```

6.5 Final predictions

For the final predictions on the test data set two set of predictions was made **predicted_for_not_missing** and **predicted_for_missing** by model 1 and model 2 respectively. They were finally combined using a for loop

```
for i in range(len(submission)):
    try:
        submission.loc[i,1] = predicted_for_not_missing.loc[i,"pred"]
    except:
        submission.loc[i,1] = predicted_for_missing.loc[i,"pred"]
```

7. Learning outcomes

1. This project was quite a good learning experience for us as we got to deal with a complex and big dataset .
2. The dataset was representative of real world data. It was quite a challenge to clean and preprocess the dataset for training different models.
3. Also dealing with bias - variance tradeoff was important here as the dataset was quite big. It was important to analyze the learning curves and find the right parameters for training a robust model.
4. The dataset also had class imbalance so we learned methods to overcome it. We had used methods such as undersampling and giving class weight to counter it.
5. Feature selection was also a crucial part of the project. As there were 47 variables it was important to remove the redundant ones and select the important features.
6. Detection and removal of outliers was another thing that increased the model performance.
7. Finding and handling missing data was important to improve the final predictions. For this we had build two models, one that works only on clean data and one that works on imputed data.