Fraudulent Transaction Detection in Credit Cards

Member 1	Manas Shukla	manas@vt.edu
Member 2	Prachi Pundir	prachipundir@vt.edu
Member 3	Anupa Shah	anupashah@vt.edu

Introduction

Credit card industry is a billion-dollar industry and for all banks, retaining high profitable customers is the number one business goal. Credit card fraud is any dishonest act and behavior to obtain information without the proper authorization from the account holder for financial gain. Among different ways of frauds, Skimming is the most common one, which is the way of duplicating of information located on the magnetic strip of the card. Apart from this, the other ways are:

- Manipulation/alteration of genuine cards

- Creation of counterfeit cards

- Stolen/lost credit cards

- Fraudulent telemarketing

Hence, we aim to detect fraudulent credit card transactions with the help of machine learning models.

Project Problem Statement

Banking fraud poses a significant threat and in terms of substantial financial losses, trust, and credibility, this is a concerning issue to both banks and customers alike. If we can detect fraudulent transactions accurately, millions of dollars can be saved that go in settling these transactions after a long period of time due to late detection or reporting. In the banking industry, credit card fraud detection using machine learning is a necessity for them to put proactive monitoring and fraud prevention mechanisms in place. Machine learning will help these institutions to reduce time- consuming manual reviews, costly chargebacks and fees, and denials of legitimate transactions. The problem is best approached using data analysis and machine learning because detecting only outliers by mere statistics is erroneous and does not fare well in solving the problem. To approach this problem, we shall deal with class imbalances, model selection and hyperparameter tuning.

Data Set

The synthetic credit card transaction dataset is obtained from GitHub :

https://github.com/IBM/TabFormer/tree/main/data/credit_card

The Dataset is also available at:

https://ibm.ent.box.com/v/tabformer-data

The initial dataset includes 24M records with 15 fields.

The columns in the dataset are:

Column Name	Description
User	Unique number
Card	Card number
Year	Year of the transaction
Month	Month of the transaction
Day	Date of the transaction
Time	Time of the transaction
Amount	Amount of the transaction
Use Chip	Describe the type of transaction
Merchant Name	Merchant Name
Merchant City	Merchant City
Merchant State	Merchant State
Zip	Merchant Zip
MCC	Merchant Category Code
Is Fraud?	Describes the transaction is fraudulent or not
Errors?	Errors during transaction

Preprocessing Steps

Data Cleaning

At first, we analyzed the data by exploratory data analysis and found that a few column names have special characters (example - '?') and few columns have null values.

```
#Checking for null values
df.isnull().sum().sort_values(ascending=False)
```

```
Errors?           23998469
Zip                2878135
Merchant State     2720821
User                     0
Card                     0
Year                     0
Month                    0
Day                      0
Time                     0
Amount                   0
Use Chip                 0
Merchant Name            0
Merchant City            0
MCC                      0
Is Fraud?                0
dtype: int64
```

a few column operations were performed to rename attribute names as listed as below:

i. All the '?' were replaced with blank characters ('') from column names.

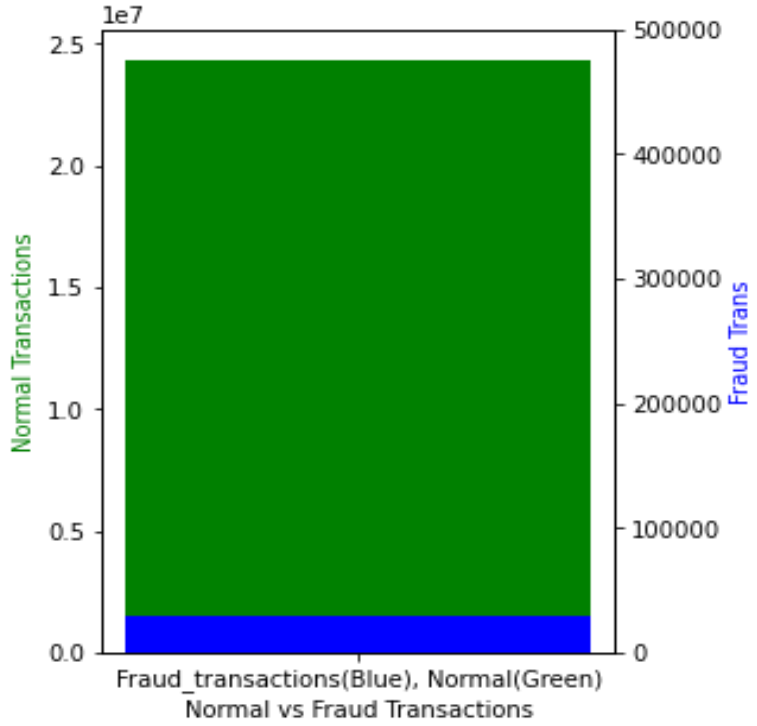
ii. All the spaces (' ') were replaced with underscores ('_')

iii. All the Merchant_States where transaction was done online were listed as NaN. Since the number of frauds was maximum in online transactions(18349 out of a total 29757 frauds), the NaN states could not be missed. Therefore, the NaN states were replaced with State name as "ONLINE"

iv. Since zip is a categorical attribute, all null values in zip were replaced by corresponding city value.

v. All null values in Errors column, were replaced by 0 as it didn't represent any error.

Now, as listed above, there is a huge difference in the number of Normal transactions and Fraud transactions. This can be seen by plotting Normal Transactions Class on a y scale of 25 million and Fraud Transactions on a y scale of 500,000.



Here Green part represents non-fraudulent transaction and blue part represents fraudulent transaction. Even though the scale of the fraud transaction plot is 50 times lesser than the normal transaction plot, the fraudulent transaction bar is still just a small fraction of the normal transactions.

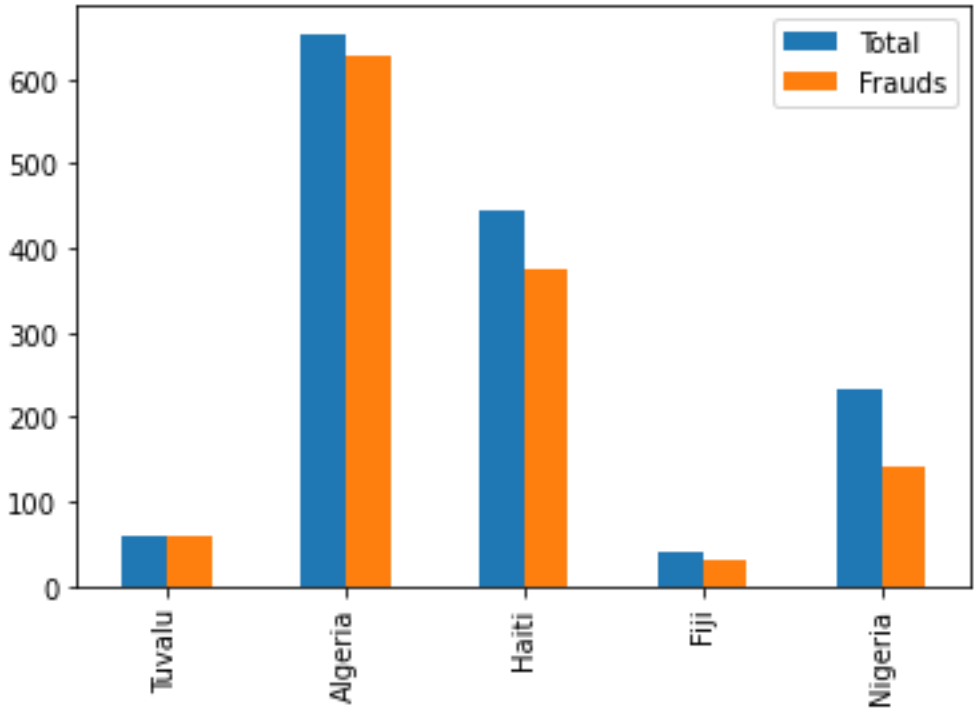
Skewness

we analyzed dataset and found frauds depend a lot on the Merchant State by calculating density of fraud in each state. It must be noted that in the dataset, states are used to denote a geographical location in general and is not restricted to a state in the United States.

Density of Fraud in a state ->

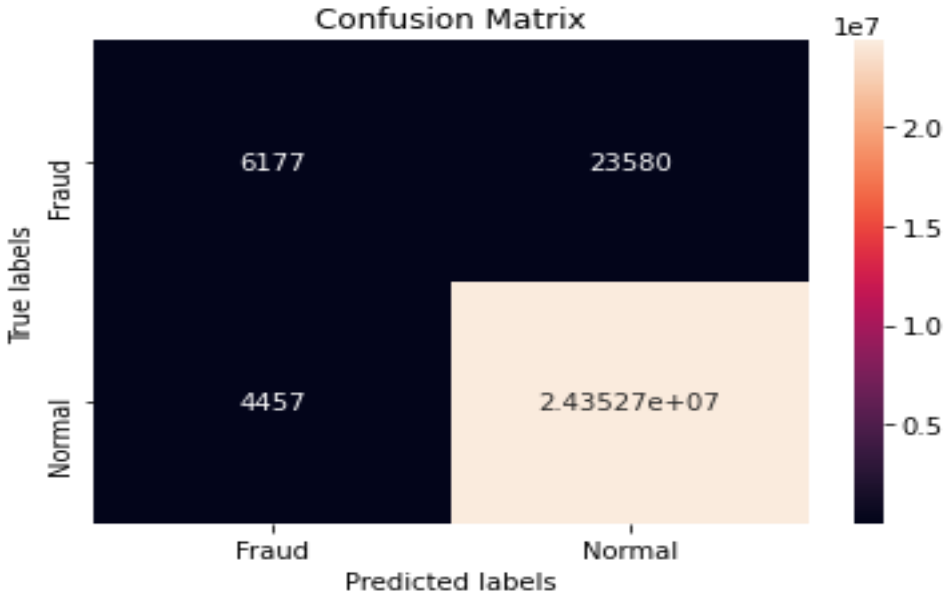
Ω = (Total Number of frauds in State) / (Total Number of transactions in a State)

Plotting 5 of the states with highest density, we get:



just by exploratory analysis, we can form a naive model that categorizes a transaction as fraudulent if it is from one of the top 7 states, and the rest of the transactions as non-fraudulent.

For this model, the following Confusion matrix was obtained:



Since the data is imbalanced, accuracy cannot be relied on so we will see the precision and recall values of this model. From the confusion matrix above, we can see that for our naive model:

Precision = 0.2076

Recall = 0.5809

F-score = 0.3059

Therefore, to build upon this model and increase its score, a stratified random sampling of the dataset by Merchant_State shall be done, after including all fraudulent transactions in each sampled dataset. The reason for including every fraudulent transaction in all the sampled datasets formed is to reduce the class imbalance while training models.

Down Sample Data – Random Stratified Sampling

The categorical attribute Merchant_States needs to be one-hot encoded since sklearn ML algorithms need categorical values to be converted into numerical values. The best way to attain an optimum model for the problem would be to one-hot encode all the categorical attributes, but that is not feasible computationally. Therefore, as seen in the Naïve model above, only Merchant_States is important to be encoded. Along with it, Card and Use_Chip can also be encoded since they don't have many unique values and encoding them would be feasible computationally. Also, the highly skewed data needs to be balanced. For this, 'undersampling the majority class' approach is adopted. Thus, five different chunks of data were formed, each containing all fraudulent transactions and all of the states in question. Random stratified sampling based on state columns was implemented after including all the fraudulent transactions so that the final chunks of dataset are representative of the original dataset. Thus, 5 different sampled datasets were obtained which were all representatives of the main dataset. The reason behind forming five chunks and not a big sampled dataset is computational efficiency. Given below is the dataframe for one of the five datasets:

	User	Card	Year	Month	Day	Time	Amount	Use_Chip	Merchant_Name	Merchant_City	Merchant_State	Zip	MCC	Errors	Is_Fraud
4099	0	0	2015	11	15	12:55	$287.13	Online Transaction	-8194607650924472520	ONLINE	ONLINE	ONLINE	3001	0	Yes
4100	0	0	2015	11	15	13:19	$2.41	Online Transaction	-7759074308363763111	ONLINE	ONLINE	ONLINE	5651	0	Yes
4101	0	0	2015	11	16	09:41	$50.81	Online Transaction	-551332107213382088	ONLINE	ONLINE	ONLINE	4411	0	Yes
4102	0	0	2015	11	16	09:46	$248.36	Online Transaction	4872340518840476610	ONLINE	ONLINE	ONLINE	5732	0	Yes
4103	0	0	2015	11	16	11:20	$473.00	Online Transaction	-8566951830324093739	ONLINE	ONLINE	ONLINE	3640	0	Yes
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
22913565	1880	6	2010	10	12	12:21	$18.35	Swipe Transaction	8263593753316459353	Andorra La Vella	Andorra	Andorra La Vella	5411	0	No
22901189	1880	1	2010	10	14	13:03	$26.22	Swipe Transaction	8263593753316459353	Andorra La Vella	Andorra	Andorra La Vella	5411	0	No
22902693	1880	2	2010	10	16	12:41	$29.29	Swipe Transaction	8263593753316459353	Andorra La Vella	Andorra	Andorra La Vella	5411	0	No
22913568	1880	6	2010	10	13	13:25	$26.13	Swipe Transaction	8263593753316459353	Andorra La Vella	Andorra	Andorra La Vella	5411	0	No
22902696	1880	2	2010	10	21	21:38	$47.64	Swipe Transaction	2714206426893853043	Andorra La Vella	Andorra	Andorra La Vella	5813	0	No

289355 rows × 15 columns

Data Cleaning

a few column operations were performed to rename attribute names as listed as below:

I. All the '$' were removed from amount column.

II. For Time, only the hour of the day is relevant for our analysis, so we keep just that

	User	Card	Year	Month	Day	Time	Amount	Use_Chip	Merchant_Name	Merchant_City	Merchant_State	Zip	MCC	Errors	Is_Fraud
4099	0	0	2015	11	15	12	287.13	Online Transaction	-8194607650924472520	45373	ONLINE	45373	3001	283908	Yes
4100	0	0	2015	11	15	13	2.41	Online Transaction	-7759074308363763111	45373	ONLINE	45373	5651	283908	Yes
4101	0	0	2015	11	16	9	50.81	Online Transaction	-551332107213382088	45373	ONLINE	45373	4411	283908	Yes
4102	0	0	2015	11	16	9	248.36	Online Transaction	4872340518840476610	45373	ONLINE	45373	5732	283908	Yes
4103	0	0	2015	11	16	11	473.00	Online Transaction	-8566951830324093739	45373	ONLINE	45373	3640	283908	Yes
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
22913565	1880	6	2010	10	12	12	18.35	Swipe Transaction	8263593753316459353	32	Andorra	32	5411	283908	No
22901189	1880	1	2010	10	14	13	26.22	Swipe Transaction	8263593753316459353	32	Andorra	32	5411	283908	No
22902693	1880	2	2010	10	16	12	29.29	Swipe Transaction	8263593753316459353	32	Andorra	32	5411	283908	No
22913568	1880	6	2010	10	13	13	26.13	Swipe Transaction	8263593753316459353	32	Andorra	32	5411	283908	No
22902696	1880	2	2010	10	21	21	47.64	Swipe Transaction	2714206426893853043	32	Andorra	32	5813	283908	No

289355 rows × 15 columns

Feature Engineering - One-Hot Encoding

To select the number of features to one-hot encode, the unique count of categorical features was analyzed.

```
In [3]: print(len(pd.unique(df['Use Chip']))),print(len(pd.unique(df['Merchant State'])))
```

```
3
224
```

In the features: Merchant State, Card and Use Chip, One-hot encoding can be used. This change transformed these columns to their integer representation.

After this, the total number of features in dataset were now 249.

```
In [22]: print(len(pd.unique(df['Use_Chip']))),print(len(pd.unique(df['Merchant_State'])))
```

```
3
200
```

Considering "Use Chip" variable as an example, there were 3 categories and therefore 3 binary variables were needed. A "1" value was placed in the binary variable for the category and "0" values for the other category.

Since the data were to have numerous rows and columns after the feature transformation, the entire data was divided into 5 different chunks, and then the chunks were transformed using One-Hot Encoding.

Below is the output of one chunk of the data frame with 249 columns. We removed unnamed

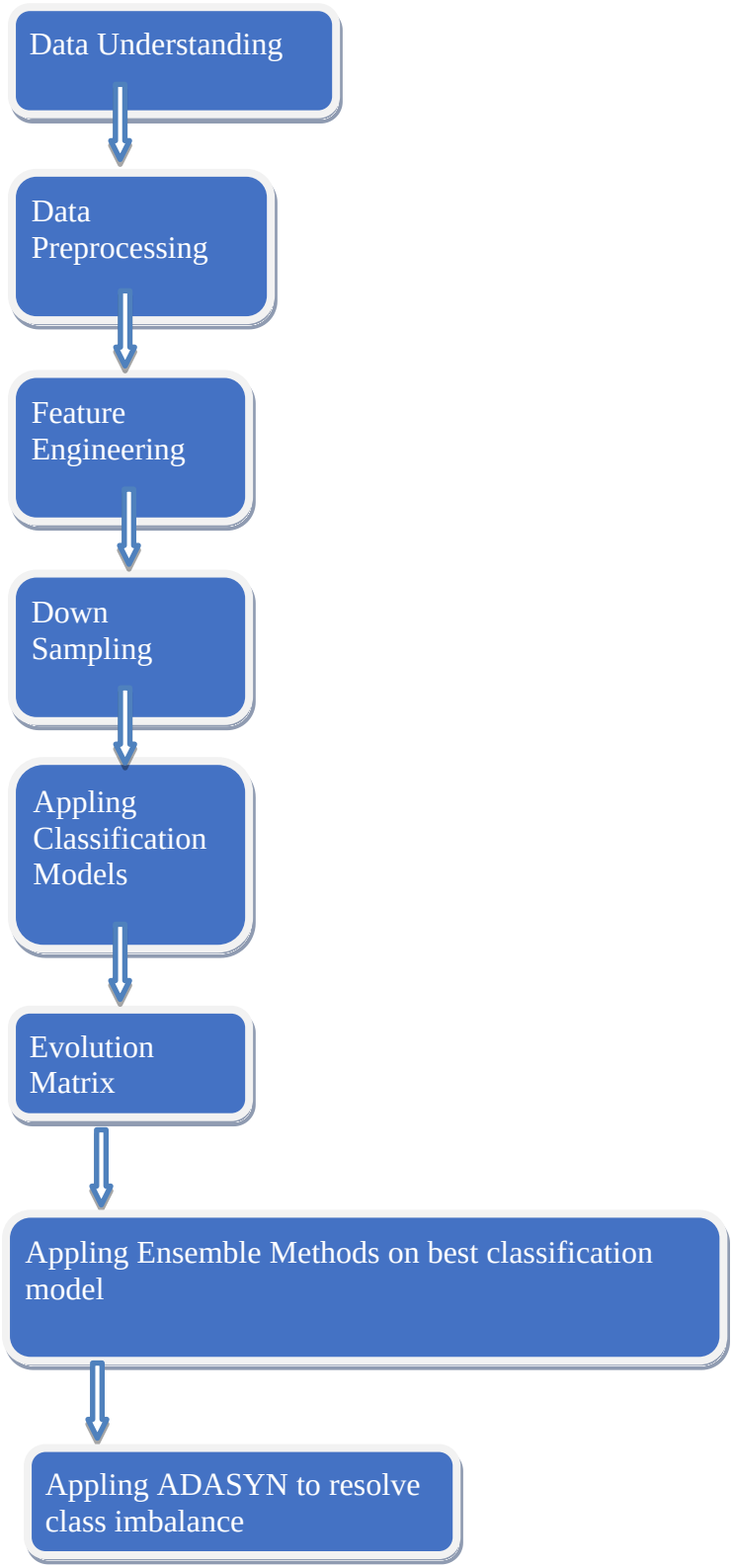
column from dataset which is not useful.

	Unnamed: 0	Merchant_State=AA	Merchant_State=AK	Merchant_State=AL	Merchant_State=AR	Merchant_State=AZ	Merchant_State=Albania	Merchant_Stat
0	0	0.0	0.0	0.0	0.0	0.0	0.0	
1	1	0.0	0.0	0.0	0.0	0.0	0.0	
2	2	0.0	0.0	0.0	0.0	0.0	0.0	
3	3	0.0	0.0	0.0	0.0	0.0	0.0	
4	4	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
575126	24386290	NaN	NaN	NaN	NaN	NaN	NaN	
575127	24386370	NaN	NaN	NaN	NaN	NaN	NaN	
575128	24386502	NaN	NaN	NaN	NaN	NaN	NaN	
575129	24386615	NaN	NaN	NaN	NaN	NaN	NaN	
575130	24386699	NaN	NaN	NaN	NaN	NaN	NaN	

575131 rows × 249 columns

Project Pipeline

Diagram



The project pipeline can be briefly summarized in the following four steps:

- **Data Understanding**: Here we upload the data and understand the features present in it. This step helped us choose the features that are required for final model.

- **Exploratory data analytics (EDA)**: In this step we performed the following:
    - Univariate and Bivariate analyses of the data
    - Feature Engineering (Column-Transformations and One-Hot Encoding).
    - Random Stratified Sampling was used since data scale is highly imbalanced and cannot be trained easily. The motive of performing this type of sampling was to ensure that the sampled data should be the representative of the original data.
    - Skewness of the data is checked and removed since some of the data points in a skewed distribution towards the tail may act as outliers for the machine learning models which are sensitive to outliers and hence that may cause a problem. Also, if the values of any independent feature are skewed, depending on the model, skewness may affect model assumptions or may impair the interpretation of feature importance.

- **Train/Test Split:** In this step train/test split is performed to check the performance of our models with seen & unseen data. And for validation, we used the k-fold cross-validation method with an appropriate k value so that the minority class is correctly represented in the test folds.

- **Model-Building/Hyperparameter Tuning:** This is the final step in which we build different models and fine-tune their hyperparameters until we got the desired level of performance. We did a comparative analysis of several machine learning techniques on each chunk.
- **Improving model performance by using ensemble methods and ADASYN to resolve class imbalance**

Methods and Models:

We evaluated Dataset on different models.

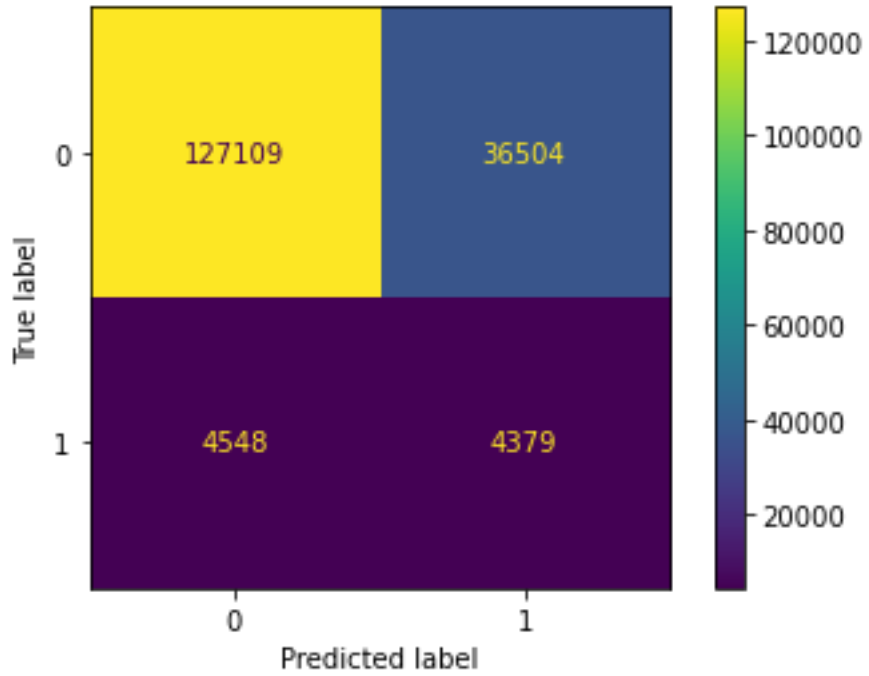
Since the data is skewed towards the Non-Fraud transactions and we are trying to maximize True Positives while minimizing False Positives, the metric used for evaluating the performance of the models shall be the f1-score, which takes both precision and recall into account. Also, the average was set to "binary" in sklearn.metrics since this is a two class classification. Setting it to weighted would result in optimistic but inaccurate metrics. The below models were first evaluated after a 5-fold cross validation and returned the metric values as mentioned in the below table:

	Recall	Precision	F1-score
Logistic Regression	0.4965 ±0.0020	0.1082 ±0.0023	0.1776 ±0.0033
Multilayer Perceptron	0.2990 ±0.2441	0.0481 ±0.0470	0.0812 ±0.0773
Decision Tree	0.8690 ±0.0027	0.8633 ±0.0052	0.8661 ±0.0034
Naive Bayes	0.9888 ±0.0008	0.1029 ±0.0017	0.1865 ±0.0027
K-Nearest Neighbor	0.7854 ±0.0027	0.9012 ±0.0047	0.8393 ±0.0015

After this, the models were predicted on the test data and further evaluated through their testing scores and confusion matrix.

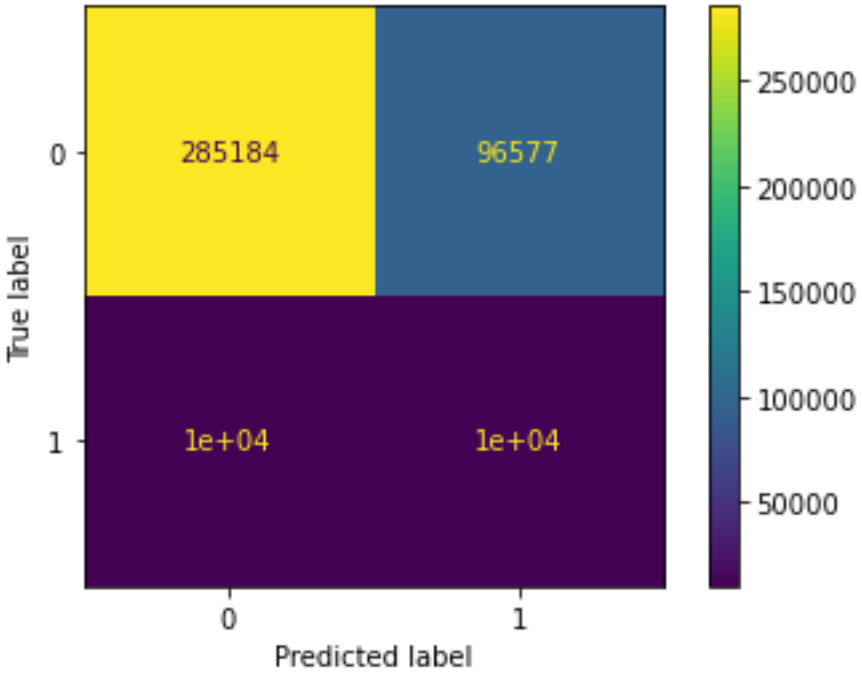
Logistic Regression

```
recall    = 0.4905
precision = 0.1071
f1        = 0.1758
```



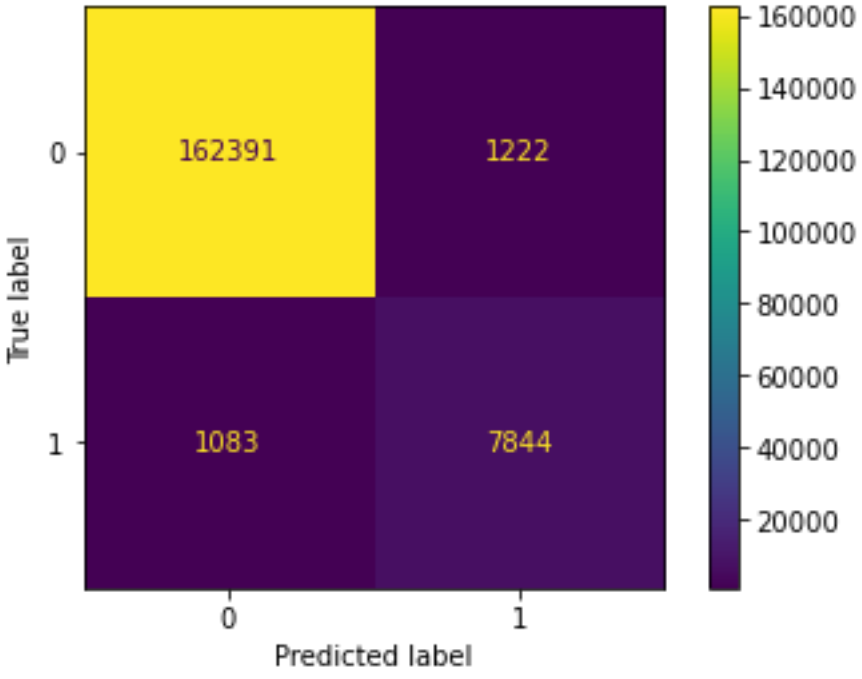
Multi-layer Perceptron

```
recall    = 0.5094
precision = 0.0993
f1        = 0.1662
```



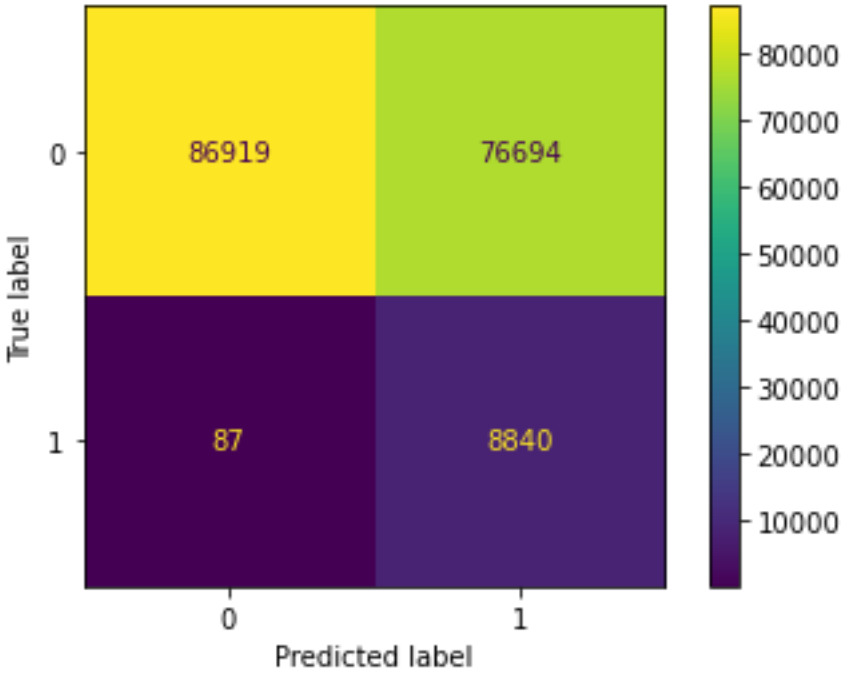
Decision Tree

```
recall    = 0.8786
precision = 0.8652
f1        = 0.8718
```



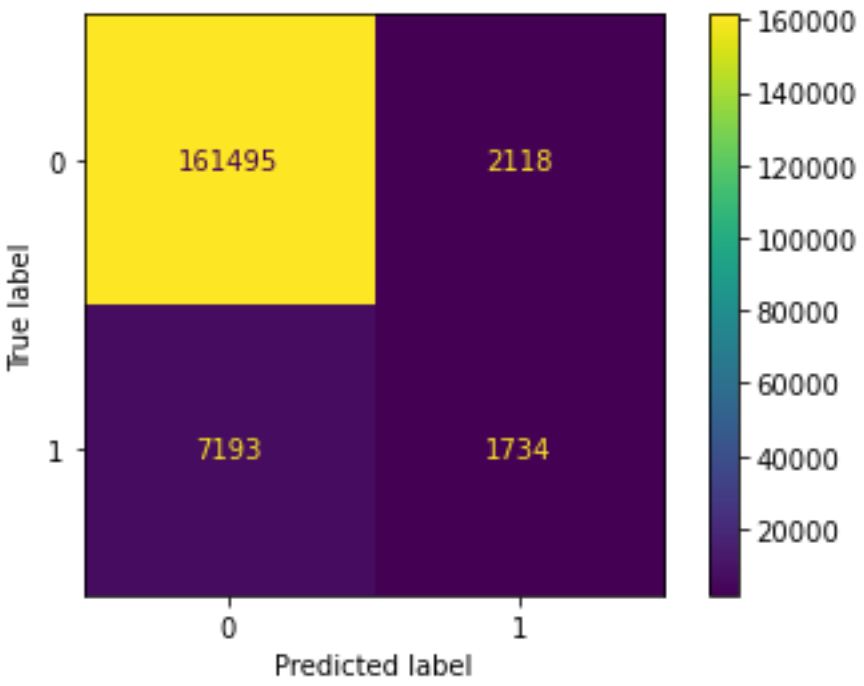
Naïve Bayes

```
recall    = 0.9902
precision = 0.1033
f1        = 0.1871
```



K-nearest Neighbour

```
recall    = 0.1942
precision = 0.4501
f1        = 0.2713
```



Insights:

The test dataset scores of only decision tree were good enough for further analysis. Therefore, it was decided to further create a better model built on decision trees. KNN also performed well for the training set but returned poor metrics for the test set. The Bernoulli Naïve Bayes classifier returned a high recall but low precision value. This is because it returned a lot of false positives as can be seen in its confusion matrix.

Improving the Best Classifier - Decision Tree

In the previous module, it was found out that Decision Trees are the best classifiers for the problem of detecting frauds. But, a classic decision tree still had a testing F1 score of 0.8718946256877674

To further improve this score, ensemble models based on Decision trees may be applied on the data. In this section, we will look at 5 different types of ensemble models for Decision trees and evaluate their best hyperparameters:

Basic Decision Tree

Bagging Classifier

Random Forest Classifier

Gradient Boosting Classifier

XG Boost Classifier

At First, For evaluating Hyperparameters, we will only use a subset of the loaded csv data for faster computation. Since we plan to use GridSearchCV for our analysis, we can expect a lot of fits and hence, a smaller chunk of data will be used to evaluate the best hyperparameters then the testing scores of each module will be seen.

We listed the best hyper parameter for each ensemble classifier based on chunk of 10000 rows.

The test scores are in order: Precision, Recall, F-1

Bagging Classifier (Bootstrap aggregating)

```
Best hyperparameters:  {'base_estimator__criterion': 'entropy',
'n_estimators': 20}

------ Test Scores: -------
[0.7841658812441094]
[0.942242355605889]
[0.8559670781893004]
```

Test score according to best hyperparameter on all data.

```
------ Test Scores: -------
[0.873865800380867]
[0.96415770609319]
[0.9167939828416969]
```

Random Forest Classifier

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best f1:  0.802349702408843
Best hyperparameters:  {'criterion': 'entropy', 'n_estimators': 200}

------ Test Scores: -------
[0.7417530631479736]
[0.8773690078037905]
[0.8038815117466803]
```

Test score according to best hyperparameter on all data.

```
------ Test Scores: -------
[0.8335387028116948]
[0.9436905516804058]
[0.8852010468712824]
```

Gradient Boosting Classifier

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Best f1:  0.8648581338447737
Best hyperparameters:  {'n_estimators': 500}

------ Test Scores: -------
[0.8152686145146089]
[0.9172852598091198]
[0.8632734530938123]
```

Test score according to best hyperparameter on all data.

```
------ Test Scores: -------
[0.8581830402150779]
[0.9567878106656675]
[0.9048068973662454]
```

XG Boost Classifier

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Best f1:  0.8651531601013968
Best hyperparameters:  {'n_estimators': 200}

------ Test Scores: -------
[0.8143261074458058]
[0.9320388349514563]
[0.869215291750503]
```

Test score according to best hyperparameter on all data.

```
------ Test Scores: -------
[0.9118404839251708]
[0.9675502198977772]
[0.9388696655132641]
```

It can be seen clearly that ensemble methods result in better performance than a sole Decision tree. But the highest score is still 0.9388696655132641. There is still scope to improve this score. We initially balanced the class after the exploratory analysis by undersampling the data. Now to further balance the classes, we use the technique of 'oversampling'. The technique used here for oversampling and generating synthetic data for the imbalanced class is ADASYN. For running this, imblearn library is required to be installed.

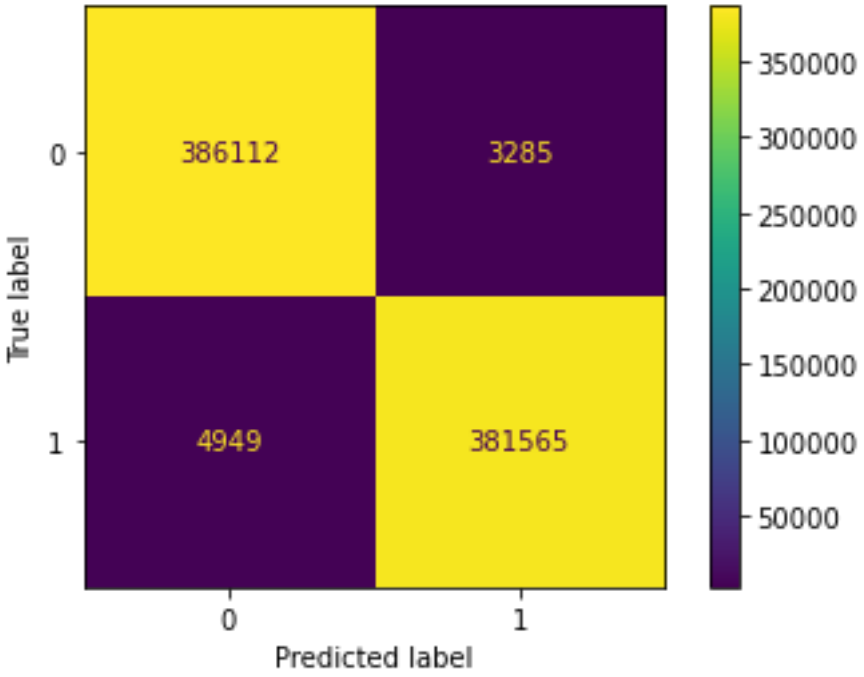
Now we evaluate the performance of all models with their best parameters on the balanced data generated by ADASYN

```
Decision Tree
------ Test Scores: -------
[0.955068975334524]
```

```
[0.976436500271225]
[0.9656345473220068]

XG Boost
[10:40:32] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was
changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like
to restore the old behavior.
------ Test Scores: -------
[0.978621528586588]
[0.991178976341831]
[0.9848602256965924]

Gradient Boosting
------ Test Scores: -------
[0.9644770556720242]
[0.9829339382509265]
[0.9736180330117508]

Random Forest
------ Test Scores: -------
[0.9599865413981417]
[0.9872898838122364]
[0.9734467984594287]

Bagged Boosting
------ Test Scores: -------
[0.9589900872220928]
[0.9909867743617861]
[0.974725917936509]
```

Therefore, after oversampling the data by applying ADASYN, it is evident that the XGBoost performs the best with an f1 score of 0.984860. Now, we need to see its performance over the entire sampled dataset with its best parameters obtained (n_estimators=200)

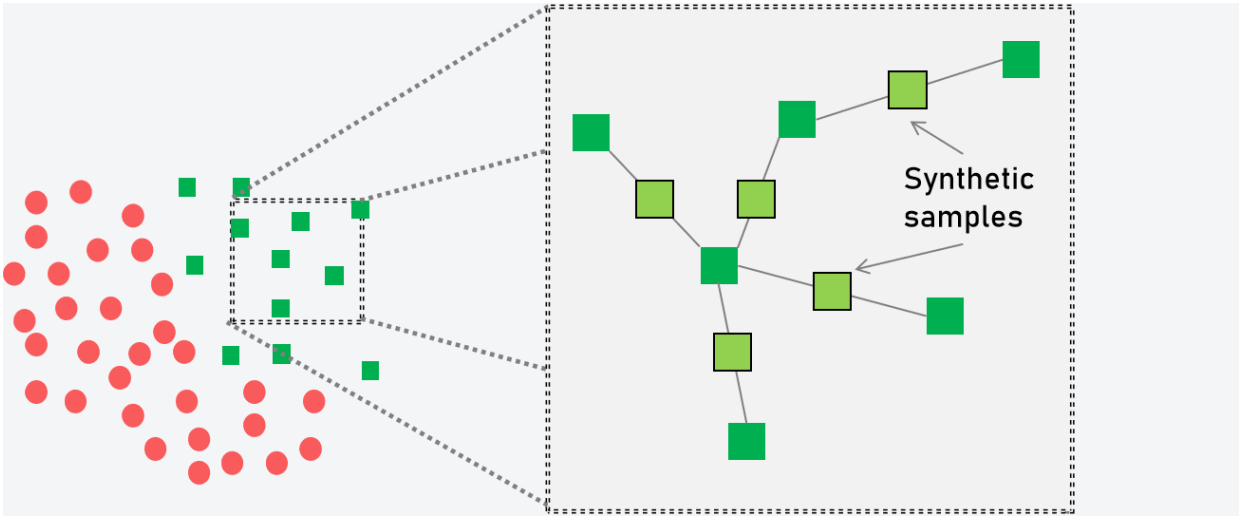
```
------ Test Scores: -------
[0.9871958066202]
[0.9914642068338314]
[0.9893254027929747]
```



Therefore, the final testing f1-score of XGBoost with ADASYN is 0.9893254027929747 or 98.933% with a precision of 98.719%

We used certain methods to mitigate this problem since further **Undersampling** could have resulted in loss of original data and **Oversampling** does not add any new information and may even exaggerate the existing information quite a bit. So we used ADASYN to resolve this problem and analyzed the performance of these techniques on our models.

- **Synthetic Minority Over-Sampling Technique (SMOTE):** In this process, you can generate new data points, which lie vectorially between two data points that belong to the minority class. These data points are randomly chosen and then assigned to the minority class. This method uses K-nearest neighbours to create random synthetic samples. The steps in this process are as follows:
  - Randomly selecting a minority point A
  - The k nearest neighbours for that data point belonging to the same are found and then a random point, B form the k_neighbours is selected.
  - Specifying a random value in the range [0, 1] as λ
  - Generating and placing a synthetic sample between the two points A and B on the vector located at λ % from the original point A.



New synthetic sample points are added in between the two homogenous class points.

- **ADAptive SYNthetic (ADASYN):** This is similar to SMOTE, with a minor change in the generation of synthetic sample points for minority data points. For a particular data point, the number of synthetic samples that it will add will have a density distribution, whereas, for SMOTE, the distribution will be uniform. The aim here is to create synthetic data for minority examples that are harder to learn, rather than the easier ones.

  To sum it up, ADASYN offers the following advantages:
  It lowers the bias introduced by the class imbalance.
  It adaptively shifts the classification decision boundary towards difficult examples.

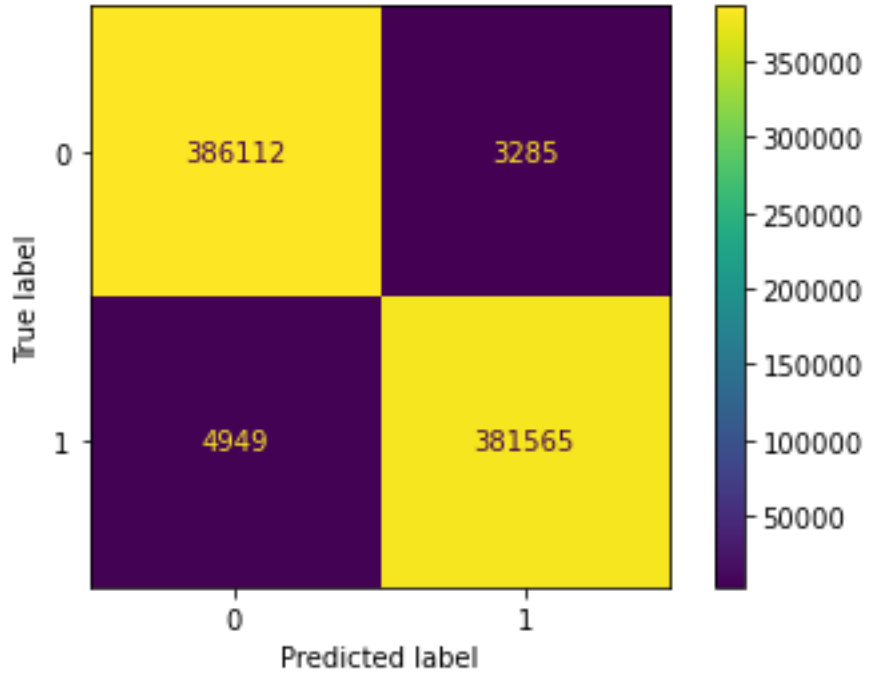
We used ADASYN in over model to improve result, since our minority example is harder to learn for most classifiers.

After applying ADASYN, the performance of the best classifier was evaluated over the entire sampled data.

Performance on the entire sampled data, using all the five chunks

Results

Training Data – 70%, Tesing Data – 30%



Therefore, the final testing f1-score of XGBoost with ADASYN is 0.9893254027929747 or 98.933% with a precision of 98.719%

Conclusion

In this project, the data massaging and pre-processing were completed. It was found that the features for analysis could be increased by One-Hot Encoding of dataset. Further, the dataset was split into five chunks of data, and a comparative analysis was performed for different Machine Learning classification models. Finally applying best ensemble technique, which is **XGBoost with n_features=200** and ADASYN with decision tree, we were able to achieve high precision score and f1 score over all other classifiers.

In this project, we created a high-precision machine learning model to detect credit card frauds. Whenever a fraud is detected, human intervention is needed to verify whether the transaction is legitimate by calling the customer. This intervention can be reduced by a high precision model, as the number of false positives will be reduced.