



Online Payments Fraud Detection

By

Aminul Islam 23341055 ,

Nishat Tabassum Srabony 23341033

April 27th, 2024

CSE422

In partial fulfillment of

With

Md. Mustakin Alam

Labib Hasan Khan

BRAC University

Merul Badda ,Dhaka

1. Introduction.....	3
2. Dataset Description.....	3
2.1 Source.....	3
2.2 Dataset Description.....	3
2.3 Imbalanced Dataset.....	5
3. Data Pre-Processing.....	5
3.1 Handling Null Values.....	5
3.2. Handling Categorical Values.....	7
4. Feature Scaling.....	8
5. Dataset Splitting.....	9
6. Model Training & Testing.....	11
6.1. KNN.....	11
6.2. Logistic Regression.....	11
6.3. Naive Bayes.....	13
6.4 Bar chart showcasing prediction of all models.....	14
6.5 Precision, recall comparison of each model.....	15
6.6 Confusion Matrix.....	16
KNN.....	16
Logistic Regression.....	17
Naive Bayes.....	18
8. Conclusion.....	19

1. Introduction

Online payment fraud is a widespread issue that has an impact on people, companies, and financial institutions all around the world. Fraudulent transactions can lead to large financial losses, reputational harm, and operational difficulties. Therefore, it is crucial to be able to accurately identify and stop fraudulent transactions. The Kaggle dataset offers a great chance to deal with this problem because it contains a variety of transactional data that can be utilized to create an effective fraud detection system model. We intend to develop a model using machine learning algorithms and methods that can correctly detect fraudulent transactions in real-time, giving insightful information that can be applied to stop additional fraud. In addition to creating a model that can precisely identify fraudulent transactions, our initiative aims to make a bigger contribution to the fight against online payment fraud. By disclosing our research and insights, we seek to increase public awareness of the problem and encourage others to come up with comparable solutions. Our ultimate goal is to contribute to the development of a more secure and safe online payment ecosystem for everyone.

2. Dataset Description

2.1 Source

- Source: Kaggle (<https://www.kaggle.com>)
- Link: <https://www.kaggle.com/datasets/kartik2112/fraud-detection>
- Reference: Kartik Shenoy (2020). Fraud Detection, Kaggle dataset

2.2 Dataset Description

In this dataset, we have a total of

- 23 features.

- Classification Problem.
- 1296675 rows of data (data points).
- 12 of the 23 characteristics in our dataset have categorical values, while the remaining 16 have quantitative values.

```
[ ] dataset1.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1296675 entries, 0 to 1296674
Data columns (total 23 columns):
 #   Column              Non-Null Count  Dtype
---  -
 0   Unnamed: 0          1296675 non-null int64
 1   trans_date_trans_time 1296675 non-null object
 2   cc_num              1296675 non-null int64
 3   merchant            1296675 non-null object
 4   category            1296675 non-null object
 5   amt                 1296675 non-null float64
 6   first               1296675 non-null object
 7   last                1296675 non-null object
 8   gender              1296675 non-null object
 9   street              1296675 non-null object
10   city                1296675 non-null object
11   state               1296675 non-null object
12   zip                 1296675 non-null int64
13   lat                 1296675 non-null float64
14   long                1296675 non-null float64
15   city_pop            1296675 non-null int64
16   job                 1296675 non-null object
17   dob                 1296675 non-null object
18   trans_num           1296675 non-null object
19   unix_time           1296675 non-null int64
20   merch_lat           1296675 non-null float64
21   merch_long           1296675 non-null float64
22   is_fraud            1296675 non-null int64
dtypes: float64(5), int64(6), object(12)
memory usage: 227.5+ MB
```

Figure 2.1 : Dataset Features

We tried to find the correlation between all the features by applying a heatmap using the Seaborn library

```
[ ] heatmap_corr= sns.heatmap (data_corr,cmap='YlGnBu') # Heatmap are used to visualize data patterns.
heatmap_corr
```

<Axes: >

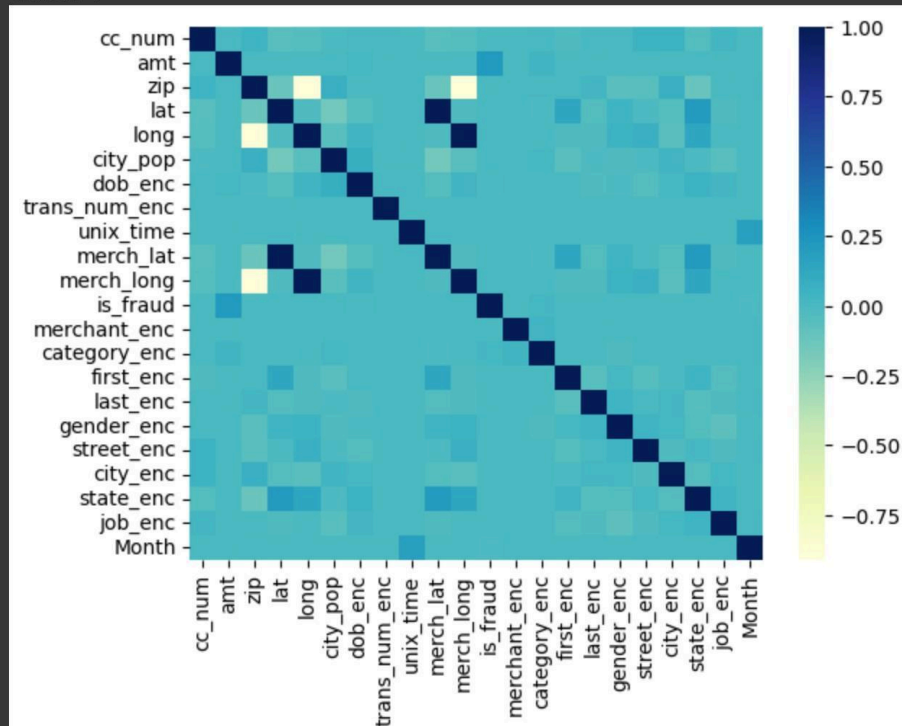


Figure 2.2 : Heatmap of the Imbalanced Dataset

2.3 Imbalanced Dataset

Our dataset contains 1296675 rows of data (data points), of which only 7506 rows are designated as fraud and 1296675 rows are designated as legitimate. Declared to be genuine. We employed one strategy in order to correct this data imbalance. which is undersampling.

3. Data Pre-Processing

3.1 Handling Null Values

The dataset under consideration required preprocessing to handle null values and optimize feature selection. To achieve this, a structured approach was adopted to maintain data integrity

and usability. As a preliminary step, a new feature named 'dummy' was introduced to the dataset. Although the purpose of this feature was not explicitly stated, its presence played a role in the subsequent preprocessing steps. The 'amt' feature was identified as having null (NaN) values. Null values can introduce biases and inaccuracies in analyses, making their treatment essential. The 'dummy' feature, which was previously introduced, was subsequently removed from the dataset. The removal was achieved using the drop method.

Before Pre-processing image

```
[ ] dataset1.isnull().sum()

Unnamed: 0                0
trans_date_trans_time      0
cc_num                    0
merchant                  0
category                  0
amt                      1308
first                     0
last                      0
gender                    0
street                    0
city                      0
state                     0
zip                       0
lat                       0
long                      0
city_pop                  0
job                       0
dob                       0
trans_num                 0
unix_time                 0
merch_lat                 0
merch_long                0
is_fraud                  0
dummy                    1296675
dtype: int64
```

After pre-processing

```
[ ] dataset1.isnull().sum()

Unnamed: 0                0
trans_date_trans_time      0
cc_num                    0
merchant                  0
category                  0
amt                      1308
first                     0
last                      0
gender                    0
street                    0
city                      0
state                     0
zip                       0
lat                       0
long                      0
city_pop                  0
job                       0
dob                       0
trans_num                 0
unix_time                 0
merch_lat                 0
merch_long                0
is_fraud                  0
dtype: int64
```

Figure 3.1: Handling Null Values

3.2. Handling Categorical Values

In the dataset we're working with, some of the features are categorical variables, which implies they only have a finite number of possible values. However, many machine learning techniques require numerical input, therefore before employing these categorical variables in our models, we must encode them into numbers. For our training, "job" is an example of a very significant attribute that is a category value. Therefore, we use Label encoder to encrypt these kinds of essential features. Categorical variables are typically encoded using label encoding, which gives each category in the variable a distinct integer value. As an illustration, suppose we have a categorical We can use a variable called color with the potential values of "red", "green", and "blue". These categories are given the labels 0, 1, and 2, correspondingly, using label encoding. A quick and simple method for transforming category variables into numerical data that may be used in machine learning models is label encoding. Because it does not require the insertion of new columns or the execution of complex data transformations, it is also relatively efficient.

```
[ ] encoded_dataset= dataset1[['cc_num','amt','zip','lat','long','city_pop','dob_enc','trans_num_enc','unix_time','merch_lat','merch_long','is_fraud','merchant_enc','category_enc','first_enc','last_enc','gender_enc','street_enc','city_enc','sta']]
data_corr= encoded_dataset.corr()
data_corr
```

	cc_num	amt	zip	lat	long	city_pop	dob_enc	trans_num_enc	unix_time	merch_lat	...	merchant_enc	category_enc	first_enc
cc_num	1.000000	0.001755	0.041459	-0.059271	-0.048278	-0.008991	0.002397	0.001415	0.000354	-0.058942	...	0.000055	0.001230	-0.027189
amt	0.001755	1.000000	0.001884	-0.001920	-0.000234	0.005796	0.010671	-0.001285	-0.000270	-0.001871	...	-0.002618	0.030847	-0.003534
zip	0.041459	0.001884	1.000000	-0.114290	-0.909732	0.078467	-0.013300	-0.000603	0.000670	-0.113561	...	0.001113	0.002371	0.004766
lat	-0.059271	-0.001920	-0.114290	1.000000	-0.015533	-0.155730	-0.049734	-0.001601	0.000632	0.993592	...	-0.002266	-0.008660	0.131869
long	-0.048278	-0.000234	-0.909732	-0.015533	1.000000	-0.052715	0.031883	0.000904	-0.000642	-0.015452	...	-0.000697	-0.000767	-0.016475
city_pop	-0.008991	0.005796	0.078467	-0.155730	-0.052715	1.000000	0.088756	0.000213	-0.001714	-0.154781	...	0.001911	0.009386	-0.066607
dob_enc	0.002397	0.010671	-0.013300	-0.049734	0.031883	0.088756	1.000000	0.000555	0.003629	-0.049343	...	0.006798	0.004347	0.004203
trans_num_enc	0.001415	-0.001285	-0.000603	-0.001601	0.000904	0.000213	0.000555	1.000000	-0.000610	-0.001571	...	-0.000459	-0.000356	0.000878
unix_time	0.000354	-0.000270	0.000670	0.000632	-0.000642	-0.001714	0.003629	-0.000610	1.000000	0.000561	...	-0.000999	0.000182	0.000483
merch_lat	-0.058942	-0.001871	-0.113561	0.993592	-0.015452	-0.154781	-0.049343	-0.001571	0.000561	1.000000	...	-0.002263	-0.008519	0.130964
merch_long	-0.048252	-0.000198	-0.908924	-0.015509	0.999120	-0.052687	0.031812	0.000934	-0.000635	-0.015431	...	-0.000673	-0.000736	-0.016489
is_fraud	-0.000981	0.219175	-0.002162	0.001894	0.001721	0.002136	-0.012156	0.000804	-0.005078	0.001741	...	-0.000536	0.020205	-0.003219
merchant_enc	0.000055	-0.002618	0.001113	-0.002266	-0.000697	0.001911	0.006798	-0.000459	-0.000999	-0.002263	...	1.000000	0.032302	-0.001576
category_enc	0.001230	0.030847	0.002371	-0.008660	-0.000767	0.009386	0.004347	-0.000356	0.000182	-0.008519	...	0.032302	1.000000	-0.001931
first_enc	-0.027189	-0.003534	0.004766	0.131869	-0.016475	-0.066607	0.004203	0.000878	0.000483	0.130964	...	-0.001576	-0.001931	1.000000
last_enc	0.006908	-0.004698	0.028295	-0.034915	-0.025557	-0.012039	-0.019168	0.000153	0.000094	-0.034548	...	-0.000439	-0.004319	-0.056983
gender_enc	0.001112	0.001013	-0.065951	0.042935	0.050404	-0.028649	-0.012185	-0.000289	-0.000960	0.042645	...	-0.000915	-0.028259	0.015197
street_enc	0.046509	0.001326	-0.053960	-0.012651	0.071328	-0.012530	-0.042514	-0.001430	-0.001089	-0.012517	...	0.000718	0.002258	-0.045369
city_enc	0.049188	-0.000567	0.074757	-0.032044	-0.066703	0.034923	0.014330	0.000798	-0.000466	-0.031828	...	-0.000294	0.000784	-0.009211
state_enc	-0.036373	0.001717	-0.116264	0.213270	0.132465	-0.012197	0.054846	-0.000183	0.001071	0.211844	...	-0.000424	0.000211	0.043145
job_enc	0.029165	-0.000623	-0.005678	-0.027593	-0.002729	-0.054945	0.028402	0.000454	0.000032	-0.027412	...	0.001360	0.003490	-0.039683
Month	-0.000281	-0.001770	0.000852	-0.001072	-0.001213	0.000137	-0.002439	-0.001285	0.184868	-0.001116	...	0.000453	0.000768	0.000162

22 rows x 22 columns

Figure 3.2 : Handling Categorical Values

4. Feature Scaling

The range of independent variables or features in a dataset can be normalized or standardized using feature scaling, a machine learning technique. The objective is for the model's predictions to be equally influenced by each characteristic. Scaling is a preprocessing technique commonly used in machine learning to normalize the features of our dataset, ensuring that they fall within a specific range. In this case, the MinMaxScaler scales features to a specified range, usually between 0 and 1.

```

sacler= MinMaxScaler()
sacler.fit(x_train)
x_train_minmaxcaler= sacler.transform(x_train)
x_train_minmaxcaler

array([[4.26924182e-05, 1.93900076e-03, 4.53711680e-01, ...,
        2.40000000e-01, 1.88640974e-01, 3.63636364e-01],
       [6.04425664e-06, 1.87405649e-03, 3.96006103e-01, ...,
        9.80000000e-01, 3.79310345e-01, 9.09090909e-02],
       [1.20411888e-03, 2.17045105e-03, 3.05535913e-01, ...,
        4.00000000e-02, 2.98174442e-01, 7.27272727e-01],
       ...,
       [6.03708219e-06, 1.97389102e-03, 2.86659295e-01, ...,
        8.60000000e-01, 8.84381339e-01, 0.00000000e+00],
       [7.57024180e-05, 1.00870875e-03, 5.32471039e-01, ...,
        9.60000000e-01, 6.04462475e-01, 1.81818182e-01],
       [7.10454961e-04, 2.03987163e-03, 2.49259635e-01, ...,
        8.60000000e-01, 8.92494929e-02, 3.63636364e-01]])

```



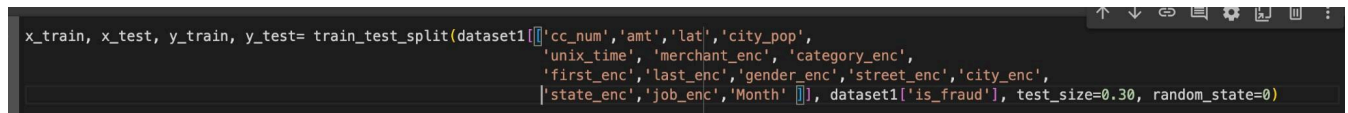
```
[ ] sacler= MinMaxScaler()
    sacler.fit(x_test)
    x_test_minmaxcaler= sacler.transform(x_test)
    x_test_minmaxcaler

array([[8.02351426e-04, 4.57321626e-03, 2.23542950e-01, ...,
        3.60000000e-01, 4.50304260e-01, 4.54545455e-01],
       [6.89619469e-05, 7.74371404e-05, 2.43334576e-01, ...,
        2.00000000e-01, 9.04665314e-01, 1.00000000e+00],
       [7.16370822e-04, 4.24982401e-03, 2.90042900e-01, ...,
        6.00000000e-02, 2.17038540e-01, 6.36363636e-01],
       ...,
       [8.43306576e-01, 8.98455203e-03, 5.12017263e-01, ...,
        7.40000000e-01, 8.98580122e-01, 0.00000000e+00],
       [7.16340741e-04, 2.44295741e-03, 4.01712589e-01, ...,
        3.00000000e-01, 3.93509128e-01, 2.72727273e-01],
       [1.20413967e-03, 5.28416296e-04, 4.02471168e-01, ...,
        9.00000000e-01, 3.42799189e-01, 3.63636364e-01]])
```

Figure 4.1 : Feature Scaling

5. Dataset Splitting

In machine learning, dividing the dataset into training and testing sets is a common method since it enables you to assess the model's performance on untried data. Stratified splitting is essential when working with datasets that are unbalanced, meaning that there are not an equal number of samples in each class. Using the Python "sklearn.model selection" module's "train test split" method, we divide the dataset into training and testing sets. The target variable (is fraud) is the only feature in the dataset that isn't included in the X variable, while the target variable is the sole feature in the Y variable. The dataset is randomly divided into training and testing sets using the train test split function, with the test size parameter indicating a 70:30 ratio (30% of the data is used for testing). The splitting is stratified because the stratify parameter is set to y. This makes sure that the percentage of fraudulent transactions is nearly the same in the training and testing sets as it is in the original dataset.



```
x_train, x_test, y_train, y_test= train_test_split(dataset1[['cc_num', 'amt', 'lat', 'city_pop',  
                                                         'unix_time', 'merchant_enc', 'category_enc',  
                                                         'first_enc', 'last_enc', 'gender_enc', 'street_enc', 'city_enc',  
                                                         'state_enc', 'job_enc', 'Month']], dataset1['is_fraud'], test_size=0.30, random_state=0)
```

Figure 5.1 : Splitting and Scaling the dataset

6. Model Training & Testing

6.1. KNN

The K-Nearest Neighbors (KNN) classification technique assigns a label to a query point based on the majority class of its K nearest neighbors after locating the K closest data points to it in the training set. Choosing the number of K neighbors we desire. After that, calculate the separation between each point in the training set and the query point. Then, depending on the estimated distance, identify the K data points that are the most near the query location. Give the query point a class based on the majority class of its K nearest neighbors. A vote on this is requested from the K neighbors. If the K value is too low, the algorithm may overfit, and if the K value is too high, it may underfit. Here is the dataset's K-Nearest Neighbors (KNN):

Model training: On the preprocessed dataset, a K-Nearest Neighbors (KNN) classifier is trained. Here, data can be imported, checked, split up, calculated for distance, predicted, and labeled.

Model evaluation: Compare the anticipated responses after the model has been tested using the same dataset that was used to train it. Comparison to other models: To determine which model performs better on this dataset, the K-Nearest Neighbors (KNN) performance is compared with that of other models like the Naive Bayes model and Logistic Regression.

KNN Accuracy
Imbalanced Dataset 99.410030
Under Sampled-Dataset 79.085258

Scores of KNN Model

6.2. Logistic Regression

In binary classification situations, where the objective is to predict a binary outcome (e.g., 1/0, yes/no, true/false) based on a set of input features, logistic regression is a statistical technique employed. Using a logistic function, it models the relationship between the input features and

the likelihood of the binary outcome. In our dataset, the logistic regression model is utilized to determine by the class label whether a transaction is real or fraudulent. Prior to using logistic regression, data must be gathered and the input attributes and binary outcome to be predicted must be determined. Then, using an optimization process that determines the best set of weights for each input characteristic that maximizes the likelihood of the observed outcomes, We fit a logistic regression model to the data.

By feeding the input features into the model once it has been trained, we may use it to forecast fresh data by receiving the expected probability of the binary outcome. The binary prediction is then obtained by applying a threshold to the expected probabilities. The logistic regression model showed very high Training and Testing Accuracy and zero Precision and Recall when applied to the original imbalanced dataset where we are solving a classification problem Precision and Recall increased after we applied Under Sampled-Dataset to our dataset, though. We have to apply Dataset Pre-Processing for this dataset in order to handle Null Values, Categorical Values, and feature scaling. Then, in order to train and test the model, we divided the dataset into training and testing sets.

By feeding the input features into the model once it has been trained, we may use it to forecast fresh data by receiving the expected probability of the binary outcome. The binary prediction is then obtained by applying a threshold to the expected probabilities. The logistic regression model showed very high Training and Testing Accuracy and zero Precision and Recall when applied to the original imbalanced dataset where we are solving a classification problem Precision and Recall increased after we applied Under Sampled-Dataset to our dataset, though.

We have to apply Dataset Pre-Processing for this dataset in order to handle Null Values, Categorical Values, and feature scaling. Then, in order to train and test the model, we divided the dataset into training and testing sets.

Model training: The preprocessed dataset is used to train the logistic regression model. Given that this dataset has a binary classification issue, logistic regression is applied. Any input value is translated into a value between 0 and 1, which reflects the expected probability of a positive event, by the logistic function, also referred to as the sigmoid function.

Model evaluation: On the test dataset, the Logistic Regression model may be assessed using performance metrics like accuracy, precision, and recall to determine how well it can distinguish between fraudulent and legitimate transactions.

Comparison with other models: To determine which model performs better on this dataset, the performance of the Logistic Regression model is compared with that of other models like K-Nearest Neighbors (KNN) and Naive Bayes. Using logistic regression on the balanced dataset after sampling reveals that it outperforms the other 2 models in terms of precision, recall, and accuracy.

Logistic Regression Accuracy
Imbalanced Dataset 99.397691
Under Sampled-Dataset 85.057726

Scores of Logistic Regression Model

6.3. Naive Bayes

Naive Bayes is employed in this dataset as a classification method to determine whether a transaction is authentic or fraudulent. A probabilistic machine learning approach called Naive Bayes, which is based on Bayes' Theorem, operates under the presumption that given the class label, the features are conditionally independent. This presumption is referred to as "naive" because it may not always be accurate, although it frequently works effectively, particularly when working with categorical or text data.

Because it is assumed that the likelihood of the features follows a Gaussian distribution, the Gaussian Naive Bayes model is employed. It works by estimating the probability of a new data point falling into each class by first computing the mean and variance of each feature for each class. The final forecast is determined by the class with the highest likelihood.

Naive Bayes revealed significantly low precision and recall for identifying fraudulent transactions when working with the original unbalanced dataset. Undersampling was used to balance the dataset, and this led to a considerable improvement in the model's performance. The model's performance before and after the dataset was balanced demonstrates the significance of correcting class imbalance when tackling fraud detection issues.

Preprocessing: The dataset is preprocessed to handle missing values, categorical variables, and to scale the features. This ensures that the algorithm can process the data efficiently. Model training: The preprocessed dataset is used to train the Gaussian Naive Bayes classifier. Since

the continuous features are assumed to follow a Gaussian (normal) distribution, which is a realistic assumption for many real-world data sets, Gaussian Naive Bayes is utilized.

Model evaluation: On the test dataset, the effectiveness of the Naive Bayes model is assessed using various measures, including accuracy, precision, recall, etc. This illustrates how accurately the model can distinguish between fraudulent and legitimate transactions.

Naive Bayes Accuracy
Imbalanced Dataset 98.717491
Under Sampled-Dataset 85.324156

Scores of Naive Bayes Model

Comparison with other models: The performance of the Naive Bayes model is compared with other models like K-Nearest Neighbors (KNN) and Logistic Regression to see which model performs better on this dataset.

Handling class imbalance dataset: To provide a balanced dataset with an equal amount of legitimate and fraudulent transactions, the dataset is undersampled. This aids in addressing the problem of class imbalance, which can lead to forecasts that are biased in favor of the dominant class.

Model training and evaluation on balanced data: To determine whether performance increases when class imbalance is resolved, the Naive Bayes model is trained and assessed once more on the balanced dataset. In terms of accuracy, precision, and recall in this particular dataset, Naive Bayes performed relatively poorly when compared to KNN and Logistic Regression. It's crucial to remember that the algorithm's performance can change based on the dataset and the particular problem being tackled.

6.4 Bar chart showcasing prediction of all models

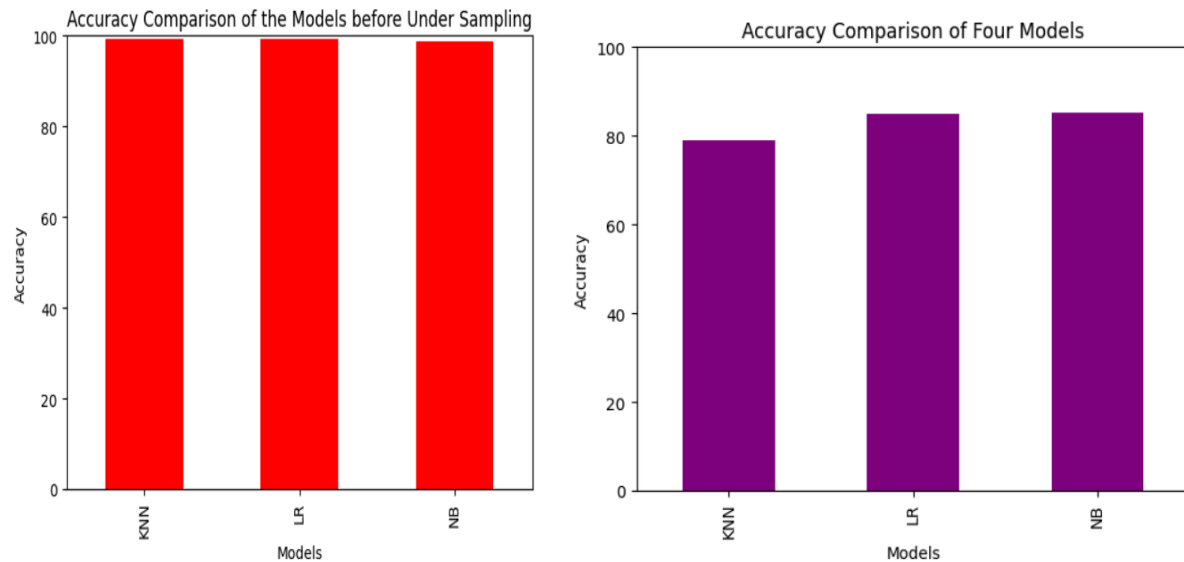
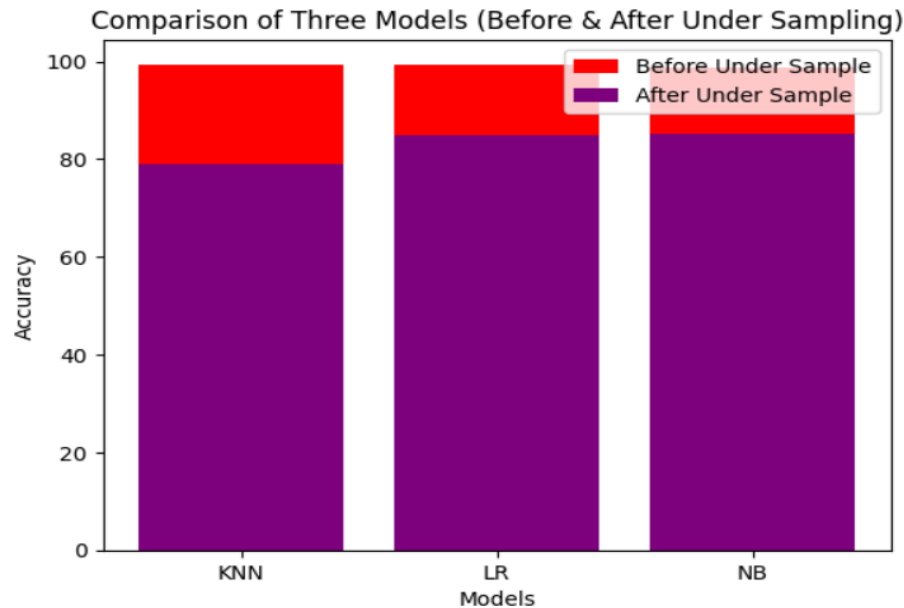


Figure: Before Undersampling

Figure: After Undersampling

6.5 Precision, recall comparison of each model

When we evaluated our models on our unbalanced dataset, they all outperformed each other, as can be shown if we compare our models based on accuracy. In comparison to the under-sampled dataset, we obtained testing accuracy for each model of around 99%, which is pretty excellent. Additionally, the undersampled dataset has a respectable accuracy rate, with k-NN scoring 75.932504 %, logistic regression scoring 83.303730 %, and naive bayes scoring 84.413854 %. Therefore, an undersampled dataset with low sampling is bad from an accuracy standpoint.



Testing accuracy scores for each model on Imbalance Dataset and Under-Sampled Dataset

6.6 Confusion Matrix

We are attempting to demonstrate and compare the confusion matrices for each model and each strategy as we trained our models using two different ways. The confusion matrix of the model trained on the unbalanced dataset is depicted in the first picture in this subsection, while the confusion matrix of the model trained on the under-sampled dataset is depicted in the second.

KNN

In the given confusion matrix, there are 386589 [TP], 2157 [FN], 138 [FP], 119 [TN]. After undersampling: 1646 TP 305 FN 637 FP 1916 TN. Comparing the two matrices, we can see that the model performed better in the second scenario in terms of detecting fraudulent transactions. This indicates a higher true positive rate (TPR) and a lower false negative rate (FNR) in the second scenario. Overall, the second model seems to have a better performance in detecting fraudulent transactions.

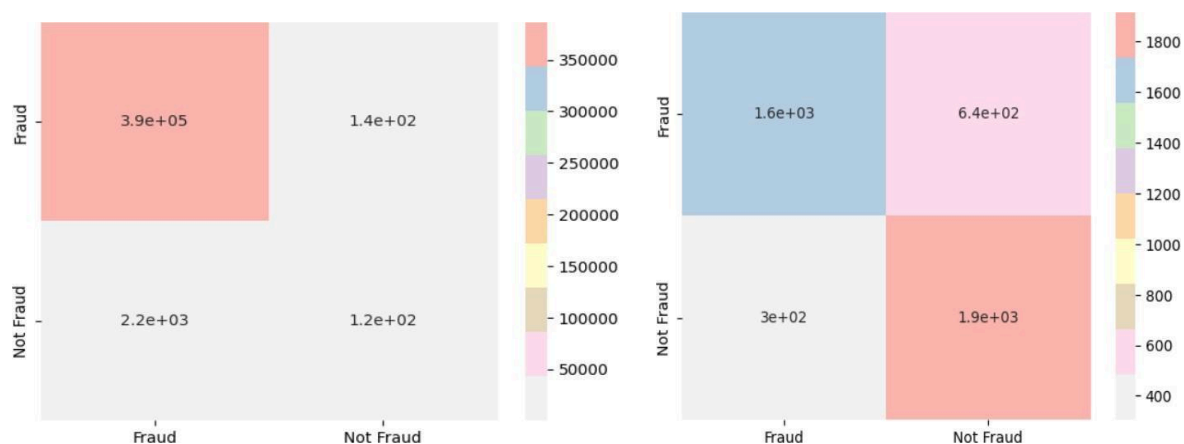


Figure: Confusion Matrix of KNN on Imbalanced Data

Figure: Confusion Matrix of KNN on Under Sampled Data

Logistic Regression

386660 [TP], 2276 [FN] 67 [FP], 0 [TN.] While for undersampling: 2131 [TP] 521 [FN] 152 [FP] 1700 [TN] We can see from a comparison of the two matrices that the second one is more effective at spotting both legitimate and fraudulent transactions. In particular, the second matrix performs better overall since it has a higher proportion of genuine positives and true negatives. The second matrix also has a greater recall for fraudulent transactions, which means that more of them are successfully recognized, and a higher specificity for genuine transactions, which means that fewer of them are mistakenly labeled as fraudulent. Overall, compared to the first model, the second one seems to be more accurate and trustworthy at spotting fraudulent transactions.

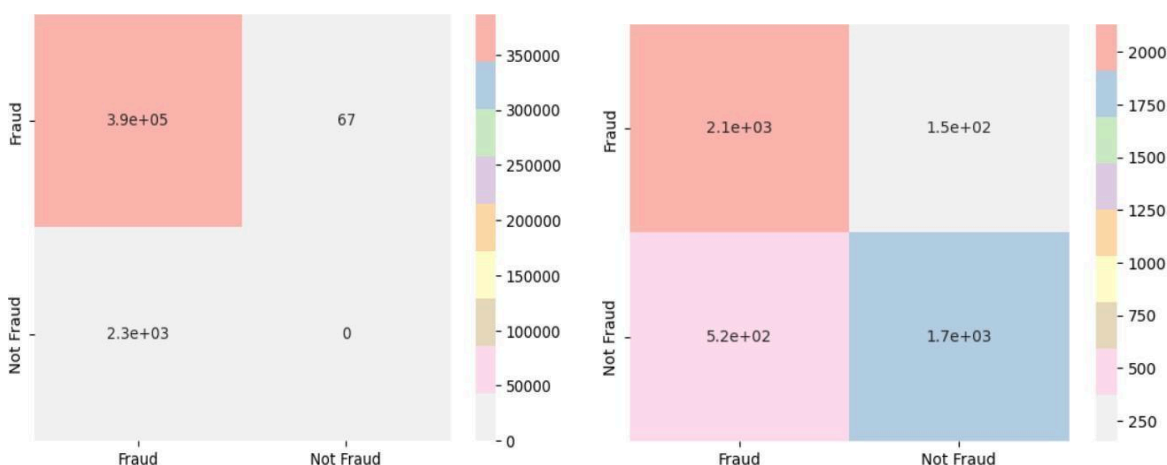


Figure: Confusion Matrix of Logistic Regression on Imbalanced Data

Figure: Confusion Matrix of Logistic Regression on Under Sampled Data

Naive Bayes

382898 TP 1160 FN 3829 FP 1116 TN. While for undersampled data: 2167 TP, 545 FN, 116 FP, 1676 TN. By contrasting the two matrices, we can see that the second one performs better in terms of identifying both legitimate and fraudulent transactions, having greater values for TP and TN. The second matrix, in particular, has a stronger recall for legitimate transactions, which means higher specificity for fraudulent transactions means that more of them are accurately identified. Fewer of them are mistakenly labeled as authentic. higher specificity for fraudulent transactions means that more of them are accurately identified. Fewer of them are mistakenly labeled as authentic.

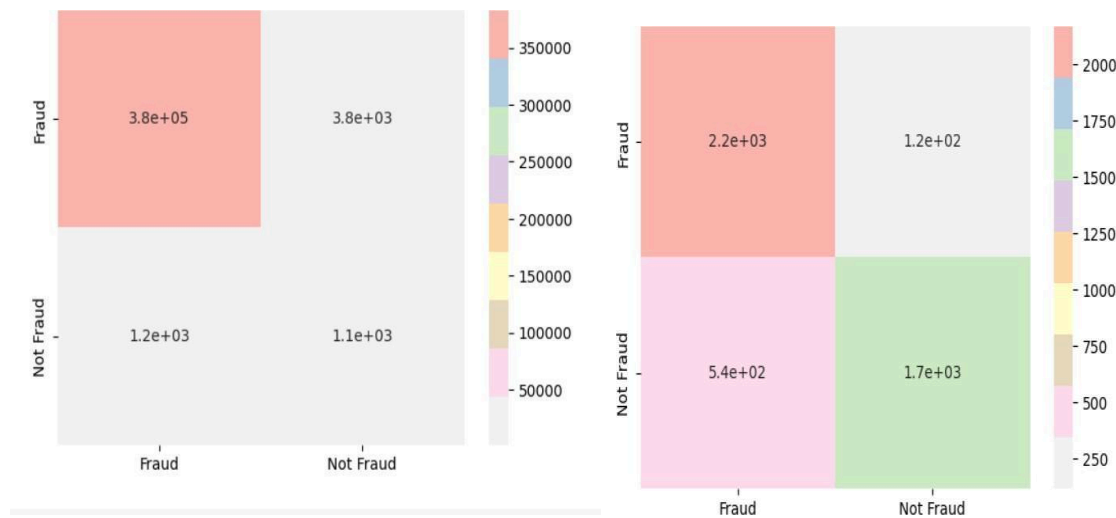


Figure: Confusion Matrix of Naive Bayes on Imbalanced Data

Figure: Confusion Matrix of Naive Bayes on Under Sampled Data

8. Conclusion

Based on the testing accuracy, precision, and recall scores of three classification models (k-NN, Logistic Regression, and Naive Bayes) on two different datasets (imbalance and under-sampled), we may conclude that KNN model works best before undersampling and the Naive Bayes model performs the best on the under-sampled dataset. It should be emphasized, however, that the under-sampled dataset is not ideal because it necessitates the elimination of a sizable amount of data, which could lead to the loss of important information. This exemplifies the problem of imbalanced datasets, when the distribution of classes is strongly skewed toward one class, making it challenging for classification algorithms to correctly identify the minority class. Future iterations of this project might incorporate other techniques for handling unbalanced datasets, like oversampling, data augmentation, and ensemble modeling. The dataset could also include additional characteristics to enhance the performance of the classification models. Finally, to determine how reliable and generalizable the models are, they may be evaluated on a larger, more varied dataset.

