Credit Card Fraud Detection

Samarth Patel Somshuvra Basu Ved Brahmbhatt

March 2023

1 Introduction

Credit card fraud is a significant problem that affects millions of individuals and businesses worldwide. As credit card usage continues to grow, so does the risk of fraudulent activity. In order to mitigate these risks, many financial institutions and businesses have turned to machine learning algorithms to detect and prevent fraudulent transactions.

The objective of this project is to develop a machine learning model that can accurately identify fraudulent credit card transactions using a dataset containing transaction data from September 2013. The dataset is highly imbalanced, with only 492 fraudulent transactions out of 284,807 total transactions, making it a challenging problem to solve.

In this report, we will discuss the steps taken to preprocess and analyze the dataset, the various machine learning models tested and their results, and the final selected model. Additionally, we will discuss the limitations of the model and suggestions for future improvements.

2 Dataset Description

1. The dataset used in this project is the Credit Card Fraud Detection dataset from Kaggle. The dataset contains transactions made by credit cards in September 2013 by European cardholders. The dataset has 284,807 rows, where each row represents a transaction, and 31 columns, where 30 columns are the anonymized features. The last column is the target variable, which is a binary variable that indicates whether the transaction is fraudulent or not.

2. The dataset is highly unbalanced, with only 492 fraudulent transactions out of 284,807 total transactions, making it a challenging problem to solve. To handle this issue, we will use various techniques such as oversampling and undersampling.

3. Before using the dataset for machine learning, we performed some preprocessing steps. We checked for null values, and fortunately, there were no missing values in the dataset. We also checked for any duplicate rows, and there were no duplicate rows in the dataset. Finally, we standardized the numerical features to have a mean of 0 and a standard deviation of 1 to improve the performance of some machine learning algorithms, that we intend to use.

3 Exploratory Data Analysis & Preprocessing

1. The first step is to check for null values present in the dataset, since non ea re present we need not do anything.

2. Next we check the distribution of data between the two classes, fraud and non-fraud, it is observed that the dataset is highly imbalanced.

3. Further we plot the respective histograms to observe the dataset and decide on how to best use the features present, the results of which are shown in the notebook.

4. Next, to preprocess the data we use StandardScalar for 'Amount' and RobustScalar for 'Time' according to our needs and then finally reshape them.

5. We then observe the the distribution of transaction time involved to get a picture of the transaction were processed, also the correlation plots are obtained to understand how the different features are inter- related.

6. Finally we do the train test split to obtain the training and test data.

4 Random Under-sampling

1. In this segment we implement Random Under-Sampling (RUS), which involves removing data with an aim of creating a balanced dataset, which in-turn prevents over-fitting of the model.
2. First we determine the degree of imbalance present in our dataset by checking the count of values present.
3. We observe there are 492 fraud cases present, so now we select 492 non fraud instances randomly from the dataset.
4. Now we define the following five models with there hypereparameters that we want to tune:
   (a) Logistic Regression
   (b) Decision Tree
   (c) Random Forest
   (d) KNN
   (e) SVM
5. GridSearch is applied to fine tune the hyperparameters, which are then used further in our models.
6. Using the best hyperparameters the following results are obtained:

Model	Accuracy	F1 score	Precision	Recall	ROC AUC
Logistic Regression	0.969945	0.095137	0.050167	0.918367	0.977827
Decision Tree	0.942909	0.051342	0.026426	0.897959	0.933778
Random Forest	0.964854	0.080808	0.042308	0.897959	0.971734
KNN	0.963537	0.078936	0.041261	0.908163	0.952608
SVM	0.964222	0.081154	0.042453	0.918367	0.977976

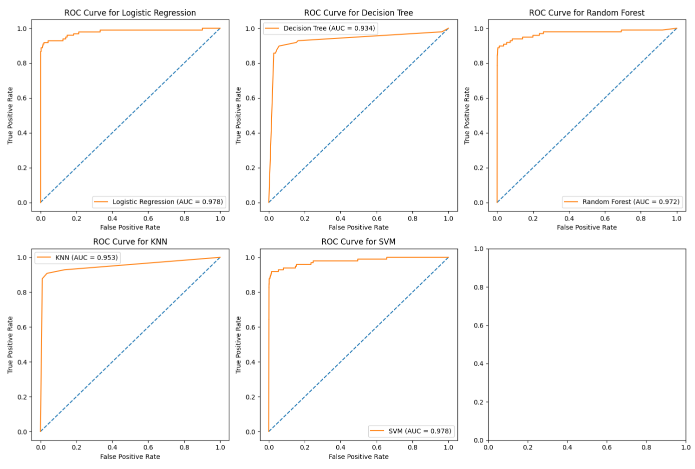

Figure 1: ROC Curves for the Models (RUS)

7. The confusion Matrix of the result before and after RUS has been shown in the notebook for all the models used.
8. The main problem with "Random Under-Sampling" is that there is a lot of information lost, which increases the possibility that our classification models will not perform as accurately as we would like them to.

5 Random Over-sampling

1. Random oversampling (ROS) is a technique used to address this issue by randomly duplicating examples from the minority class until it is balanced with the majority class. This can be done with replacement or without replacement, depending on the specific implementation. The resulting dataset will have an equal number of examples from both classes.

2. The main advantage of random oversampling is that it is a simple and effective way to improve the performance of machine learning algorithms on imbalanced datasets. By increasing the number of examples in the minority class, the algorithm will be able to learn more about this class, leading to better detection of fraud.

3. By repeating the Hyperparameter tuning steps as in the the previous section we obtain the following results:

Model	Accuracy	F1 score	Precision	Recall	ROC AUC
Logistic Regression	0.975615	0.114723	0.061183	0.918367	0.972009
Decision Tree	0.998999	0.710660	0.707071	0.714286	0.856888
Random Forest	0.999561	0.860335	0.950617	0.785714	0.962845
KNN	0.999052	0.756757	0.677419	0.857143	0.943672

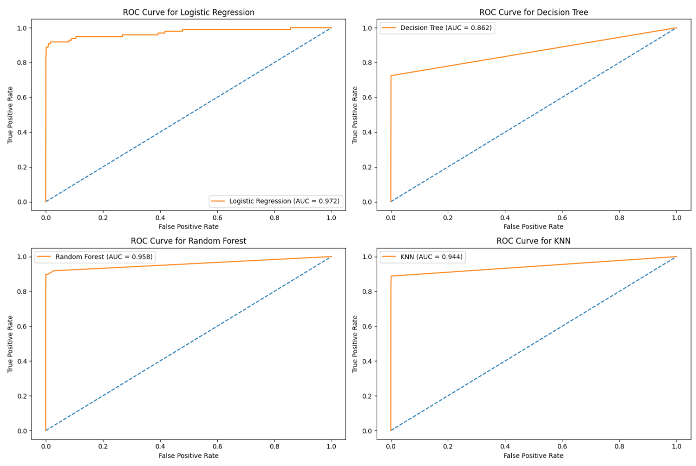

Figure 2: ROC Curves for the Models (ROS)

4. The confusion Matrix of the result before and after ROS has been shown in the notebook for all the models used.

6 SMOTE

1. In an imbalanced dataset, the number of samples belonging to one class (in this case, non-fraudulent transactions) greatly outweighs the number of samples in the other class (in this case, fraudulent transactions). This can lead to biased models that perform poorly on the minority class.

2. Synthetic Minority Over-sampling Technique (SMOTE) is a technique that creates synthetic examples of the minority class by creating new data points that are similar to the existing minority class samples, but with slight variations. Specifically, SMOTE works by selecting a minority class sample, finding its k nearest neighbors, and creating new synthetic samples along the line segments that connect the minority sample to its k nearest neighbors. By creating new synthetic samples, SMOTE increases the number of minority class samples, which can help balance the dataset and improve the performance of classification models on the minority class.

3. SMOTE can be used to create synthetic fraudulent transactions that are similar to existing ones, thereby increasing the number of fraudulent transactions in the dataset. This can help the classification model to better learn the patterns and characteristics of fraudulent transactions, and ultimately improve its accuracy in detecting fraud.

4. By repeating the Hyperparameter tuning steps as in the the previous section we obtain the following results:

Model	Accuracy	F1 score	Precision	Recall	ROC AUC
Logistic Regression	0.974457	0.110092	0.058556	0.918367	0.970103
Decision Tree	0.997665	0.539792	0.408377	0.795918	0.896966
Random Forest	0.999386	0.816754	0.838710	0.795918	0.958588
KNN	0.998086	0.612100	0.469945	0.877551	0.948482

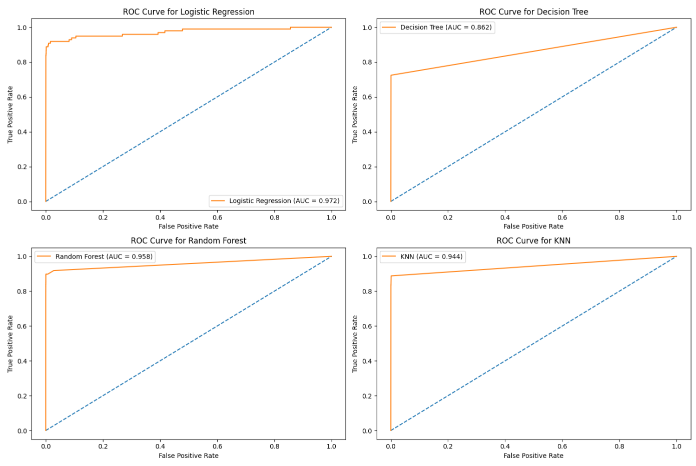

Figure 3: ROC Curves for the Models (SMOTE)

5. In this particular section SVM has not been used as it had a runtime exceeding 2 hours and was giving unsatisfactory output at the end.

6. The confusion Matrix of the result before and after application of SMOTE has been shown in the notebook for all the models used.

7 NearMiss Under-Sampling

1. NearMiss works by selecting samples from the majority class (non-fraudulent) that are close to the minority class (fraudulent) based on distance measures such as Euclidean distance. There are three versions of NearMiss: NearMiss-1, NearMiss-2, and NearMiss-3.

    (a) NearMiss-1 selects samples from the majority class that are closest to the samples in the minority class.

    (b) NearMiss-2 selects samples from the majority class that have the largest average distance to their k-nearest neighbors in the minority class.

    (c) NearMiss-3 is a modified version of NearMiss-2 that selects samples from the majority class based on the farthest distance to their k-nearest neighbors in the majority class.

2. By removing samples from the majority class, NearMiss helps to reduce the imbalance in the dataset, which can improve the performance of machine learning models in detecting fraud.

3. By repeating the Hyperparameter tuning steps as in the the previous section we obtain the following results:

Model	Accuracy	F1 score	Precision	Recall	ROC AUC
Logistic Regression	0.666567	0.009595	0.004822	0.938776	0.936615
Decision Tree	0.079456	0.003724	0.001865	1.000000	0.549618
Random Forest	0.072153	0.003657	0.001832	0.989796	0.928125
KNN	0.665040	0.009654	0.004852	0.948980	0.889770
SVM	0.185053	0.004162	0.002085	0.989796	0.898782

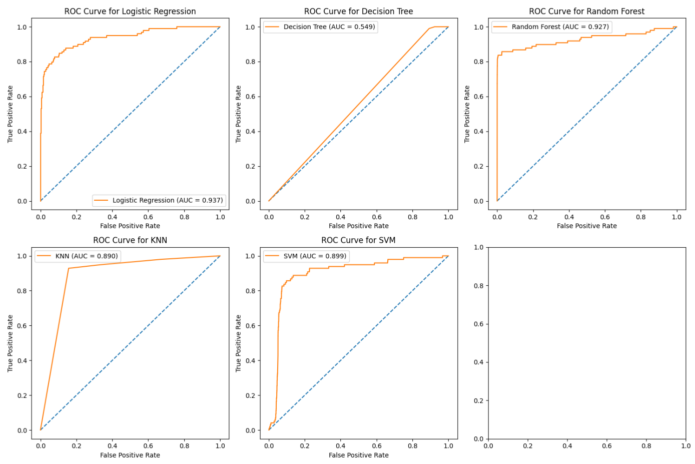

Figure 4: ROC Curves for the Models (NearMiss Under-Sampling)

4. The confusion Matrix of the result before and after application of NearMiss Under-sampling has been shown in the notebook for all the models used.

8 Analysis of the Best Models

1. We now select the best models from each technique used, by analysisng their performance metrics.

2. Next we store the metrics of the best models obtained in anew dataframe named df_best_models.

3. The following are the best results obtained so far:

Model	Accuracy	F1 score	Precision	Recall	ROC AUC
KNN	0.963537	0.078936	0.041261	0.908163	0.952608
Random Forest	0.999561	0.860335	0.950617	0.785714	0.962845
Random Forest	0.999386	0.816754	0.838710	0.795918	0.958588
Random Forest	0.072153	0.003657	0.001832	0.989796	0.928125

9 Applying LDA to the Best Models

1. In this section we implement Linear Discriminant Analysis (LDA) on the best performing models identified earlier. LDA is a technique used for dimensionality reduction, which transforms the data into a lower-dimensional space that preserves the most important discriminatory information.

2. The code first creates an LDA object and fits it on the features (X) and labels (y). It then transforms the features (X) into the LDA space using the transform() method. The transformed data is split into training and testing sets using train_test_split().

3. Next, four different sampling techniques are applied to the training set to deal with the class imbalance problem. RandomUnderSampler(), RandomOverSampler(), SMOTE(), and NearMiss() are used to balance the dataset.

4. After resampling the training set, the code defines four models - Random Forest Classifier with Random Under Sampling, Random Forest Classifier with Random Over Sampling, Random Forest Classifier with SMOTE, and Logistic Regression with NearMiss. These models are fit on the resampled training data, and their predictions are tested on the transformed testing data.

5. The accuracy, precision, recall, f1 score, and ROC-AUC score are computed for each model, and the results are stored in a dictionary called 'results_lda'. This dictionary is then used to create a pandas DataFrame called 'results_lda_df', which shows the performance metrics of each model.

6. Finally, the 'df_best_models' DataFrame is combined with the 'results_lda_df' DataFrame to create a new DataFrame called 'df_best_models_lda', which includes the performance metrics of each model both before and after applying LDA.

7. The results after applying LDA is as follows:

Model	Accuracy	Precision	Recall	F1 Score	ROC-AUC
Random Forest_RandomUnderSampling_LDA	0.921790	0.020048	0.928571	0.039250	0.976691
Random Forest_RandomOverSampling_LDA	0.999070	0.714286	0.765306	0.738916	0.928154
Random Forest_SMOTE_LDA	0.928338	0.020693	0.877551	0.040433	0.946120
Logistic Regression_NearMiss_LDA	0.999280	0.776699	0.816327	0.796020	0.981732

Table 1: Results of LDA-based models

8. The updated best models dataframe is as follows:

Table 2: Model Performance

Model	Accuracy	F1 Score	Precision	Recall	ROC AUC
KNN	0.964	0.079	0.041	0.908	0.953
Random Forest	0.999	0.860	0.951	0.786	0.963
Random Forest	0.999	0.817	0.839	0.796	0.959
Random Forest	0.072	0.004	0.002	0.990	0.928

10 Conclusion

The techniques used above had different pros and cons to them, based on how they are designed, they addressed the issue of a highly imbalanced dataset where the non-fraudulent data-points greatly extended that of fraudulent ones, which created the problem of overfitting due to the imbalance present.

1. Random Forest model performed the best in terms of accuracy, F1 score, precision, recall, and ROC-AUC across all the sampling techniques.
2. The results show that the Random Oversampling technique gave the best results in terms of accuracy, F1 score, and precision for the Random Forest model.
3. However, the Random Undersampling technique resulted in a significant reduction in the number of false positives but at the cost of higher false negatives.
4. SMOTE oversampling showed good performance in terms of recall but was not able to improve precision much.
5. NearMiss oversampling resulted in a high precision rate but relatively lower recall rate.
6. Overall, the choice of the sampling technique depends on the specific requirements of the problem, and a combination of multiple techniques could also be explored to achieve better results.
7. We also applied LDA on the best performing models, and the results showed a slight improvement in the performance metrics for Random Forest models with random oversampling and SMOTE oversampling. However, for the Random Undersampling technique, LDA did not show any significant improvement in the performance metrics. Overall, the choice of sampling technique along with LDA can further improve the performance of the models for credit card fraud detection.