Credit Card Fraud Detection Using Machine Learning

CS 534 - Artificial Intelligence
Worcester Polytechnic Institute, Fall 2023

Christina Berthiaume
MS Data Science

Nathaniel Itty
BS Computer Science
BS Data Science

Owen Radcliffe
BS Computer Science
MS Data Science

Sheroz Shaikh
MS Data Science

Abstract: As digital transactions become increasingly prevalent, the risk of credit card fraud has grown substantially, necessitating the development of robust and effective fraud detection systems. This paper presents a comprehensive study that evaluates and compares the performance of four prominent machine learning methods, Logistic Regression, K-Nearest Neighbors (KNN), Naive Bayes, and Support Vector Machine (SVM), in the context of credit card fraud classification. The dataset we are using to train these models contains information on over 2.4 million transactions, including information on the card and the user. Since most of the transactions are not fraudulent, we have a very imbalanced dataset. Therefore, we first resample the data before training our classification models. Next, we train our four machine learning models and evaluate them, choosing the best model for our goal. In this case, it is important that we minimize the number of false negatives, or cases where the model predicts that there is no credit card fraud when there actually is, because we want to correctly classify all the instances of fraud. We will choose the model that not only has a good accuracy, but also has good sensitivity.

Keywords: Classification, Imbalanced dataset, Undersampling, Logistic Regression, K-Nearest Neighbors, Naïve Bayes, Support Vector Machine, Credit Card Fraud, Ensemble methods, Bagging, Boosting, Stacking

1. Introduction

1.1 Motivation and Background

As the financial industry moves increasingly cashless, instances of credit card fraud are increasing, with the total amount of money lost to credit card fraud worldwide in 2021 projected at $32.34 billion USD, and $397.4 billion USD projected to be lost to fraud from 2022-2032 (Egan, 2023). Credit card fraud includes unauthorized transactions, identity theft, and phishing. In their 2021 annual Consumer Sentinel Network Data Book, the Federal Trade Commission noted credit card fraud as the second largest source of identity theft in the United States, with almost 390,000 reports in 2021 alone (Egan, 2023). Given the extensive financial losses incurred through credit card fraud, a reliable and accurate method of detecting fraudulent transactions is an important area of concern for many financial companies. Banks, payment processors, and merchants are constantly looking for ways to introduce new ways to use credit cards to incentivize users to make more transactions. The emergence of new technologies has paved the way for innovative payment solutions, with many people having smart devices, mobile wallets, and in-app payments (Cherif et al., 2023). However, the expansion of this technology and the use of credit cards has led to more sophisticated schemes to steal users' information, from phishing, theft, and making fake cards leading to the increase in fraudulent transactions. Cash as a method of payment has been further phased out by the COVID-19 pandemic, bringing the rise of contactless methods and making credit card payment the main representative of commerce within the global economy (Cherif et al., 2023). Artificial Intelligence is essential for designing new methods to detect cases of credit card fraud, aiming to be more proactive at preventing fraud from occurring and minimizing the number of falsely declined transactions. Fraud detection is a classification problem, identifying suspicious transactions by putting them into one of two classes: legitimate and illegitimate. Fraudulent transactions can also take two forms, online and offline. For online transactions, the culprit makes online purchases with stolen information or accounts, while offline malicious transactions are made using stolen or lost credit cards (Cherif et al., 2023). The goal of this project is to investigate four different machine learning approaches, comparing their performance in detecting fraudulent transactions, to identify the method with the highest performance in this domain.

1.2 State-of-the-Art Methods

We will be exploring four different classification methods in this paper: Support Vector Machine (SVM), Logistic Regression, K-Nearest Neighbors, and Naïve Bayes. Support Vector Machine is a machine learning algorithm that is best suited for binary classification, making it a strong choice for fraud detection. Support Vector Machine aims to find an optimal hyperplane that separates data into distinct classes. In the context of fraud detection, SVM is effective in creating a boundary that distinguishes between legitimate transactions and instances of fraud (Poongodi and Kumar 2021). It learns the behavior of fraud and genuine transactions and it then classifies the new transaction as to which it belongs. SVM’s versatility in dealing with datasets where classes may not be linearly separable is advantageous to classification tasks such as fraud detection where the input features are complex. By transforming the input data from a higher-dimensional space, SVM can effectively establish decision boundaries that maximize the margin between classes and accurately determine which transactions are fraudulent (Kumar et al., 2020). Assigning weights to transactions can also help improve the performance of the SVM algorithm and catch extreme cases.

The logistic regression model is a classification model that predicts the probability of a discrete outcome given an input variable. It is used due to its efficiency in isolating and classifying binary data, making it well-suited for identifying fraudulent transactions (Mishra & Pandey, 2021). Logistic regression is easier to implement than linear regression and more efficient to train. It makes no assumptions about the distribution of classes in the feature space, and can easily be extended to multiple classes (Alenzi & Aljehane, 2020).

K-Nearest Neighbors (KNN) is another machine learning algorithm that has found its place in the realm of credit card fraud detection. KNN is a non-parametric and instance-based algorithm. It operates on the principle that similar data points tend to belong to the same class (Chung & Lee, 2023). Unlike some other algorithms, KNN doesn't make strong assumptions about the underlying data distribution, making it well-suited for handling complex and

non-linear patterns often associated with fraud. One of the advantages of KNN is its simplicity and ease of implementation. It doesn't require a complex training process since it memorizes the entire dataset. However, this can also be a drawback when dealing with large datasets, as it can be computationally expensive to calculate distances for every data point (Alammar & Al Moayed, 2022).

Naïve Bayes is a statistical approach based on Bayesian theory that makes decisions based on the greatest likelihood (Ogundokun et al., 2021). It has the ability to make predictions for multiple cases at once (Khatri et al., 2020); in this case of credit card fraud, there are two classes: fraudulent and legitimate. In this method, the conditional independence assumption is used to ensure that the data features are independent of each other (Ogundokun et al., 2021). It is also assumed that each feature makes an equal contribution to the output (Khatri et al., 2020). These assumptions are what give the method its name because it naïvely assumes that the features are independent (Russell, 2021). Naïve Bayes has an advantage over other classification methods because it does not require as much training data (Khatri et al., 2020). This makes it more efficient and easy to understand than other methods. It is also highly accurate since the method is based on probabilities. However, there is still a risk of overfitting due to the imbalanced nature of credit card fraud data. To address this issue, we can use a variety of resampling methods which we will explore later in the paper.

1.3 Our Solution

Our proposal for this project is to train the four different classification models listed previously on credit card transactions data and evaluate the performance of these models in order to determine the best model for classifying credit card fraud. Since most credit card transactions are not fraud, we must first balance our dataset to prevent overfitting. We will use resampling methods to do this, which we will explore later in this paper. Next, we will choose the features that are most correlated with our target variable, credit card fraud, and then we will build a model using those selected features. We will create multiple different models with multiple different sets of features, and compare the performance of these models. We will compare the best versions of the four classification methods chosen using multiple different evaluation metrics. Our proposed workflow will be described in detail in Section 3.

After training the models we will evaluate their performance on a test dataset. A confusion matrix is an effective benchmark for analyzing how the models perform. The confusion matrix is formed based on true positives, positive records that are correctly labeled by the classifier, true negatives, negative records that are correctly labeled, false positives, negative records that are incorrectly identified as positive, and false negatives, positive records that are mislabeled as negative (Alenzi & Aljehane, 2020). For each model, the accuracy can be determined using the recognition rate, adding up the true positives and true negatives, and dividing it by the total number of data points. A higher accuracy will correspond to a stronger model output. Another metric to evaluate the models is sensitivity (recall), which is the true positive recognition rate. It is calculated by dividing the number of true positives by the total number of positives in the dataset. A high sensitivity is important and supplements the comparison of accuracy because a good model will correctly identify the highest number of fraudulent transactions. The maximum value is 1, which corresponds to the proportion of true positive cases equaling the number of actual positive cases (Alenzi & Aljehane, 2020). The precision of the models can also be compared, which is the percentage of identified fraudulent transactions that were actually fraudulent.

Since we are using imbalanced data, the accuracy and precision metrics alone will not be enough to evaluate the models. We can also take a look at the area under the ROC curve (AUC) to determine how well predictions are ranked, rather than their absolute values (Ogundokun et al., 2021). The ROC curve is plotted with the true positive rate on the y-axis and the false positive rate on the x-axis. A large area under the curve indicates that the model is accurate at classification (Sasank, 2019). Finally, it is not enough to evaluate the accuracy of the models. We also want our model to be efficient. To evaluate the efficiency of the model, we will take a look at two metrics: the time taken to train the model and the time to make a prediction on the testing dataset. We want to minimize both of these metrics while also optimizing the accuracy of the model.

The dataset we are using for training is a synthetic dataset of credit card transactions created by IBM which we found on Kaggle (Altman, 2019). There are three datasets: transactions, users, and cards; we plan to combine these three datasets into one. The transactions dataset consists of 24,386,900 records. There are 18 features in the users dataset, 13 features in the cards dataset, and 15 features in the transactions dataset for a total of 45 features in the

combined dataset. Features included in the user dataset are name, age, address, income, debt, FICO score, and more. The cards dataset contains features such as card type, brand, expiration date, credit limit, and date the account was opened. The transactions dataset contains information such as the date, time, amount, merchant location, merchant category, whether or not there are errors, and our target, whether or not the transaction was fraudulent.

In this report, we will first go more in-depth into our four chosen models, then detail our proposed solution and pipeline, including our data preparation, model training and testing, hyperparameter tuning, and performance measurements. Then we will discuss the preliminary results of our baseline models, the performance improvements from hyperparameter tuning, and the ensemble methods we tried to further optimize our models to predict instances of credit card fraud. After summarizing some of the lessons we have learned through our work, we present our conclusions and areas of investigation for future work.

2. SOTA Literature Review

2.1 Support Vector Machine

Support Vector Machine (SVM) is a supervised machine learning algorithm that is primarily used for classification problems. More specifically, Support Vector Machine is used for binary classification, when an attribute can be separated into two distinct classes. The classification problem in SVM is characterized as considering a training data set $\{x_1,y_1\}$, $\{x_2,y_2\}$,..$\{x_n,y_n\}$ in which i x is a vector of d dimensions, and y is a scale $\in \pm \{ 1, 1\}$ (Zhang et al., 2020). The main objective of SVM is to divide the dataset into groups to find a maximum marginal hyperplane by iteratively generating hyperplanes which segregate the classes and select the best one (Kumar et al., 2020). Support Vector Machine is based on the concept of decision hyperplanes in two dimensions that define decision boundaries for the different class memberships. A new incoming point is classified by the side of the hyperplane it falls on the vector space (Poongodi and Kumar 2021). In real world datasets, the classes cannot always be separated clearly, so a soft margin is introduced which allows some data to be classified on the wrong side of the hyperplane (Kumar et al., 2020). In the context of identifying credit card fraud, SVM will create a boundary that effectively separates legitimate transactions from instances of fraud. Two primary methods have been used to classify credit card fraud, Support Vector Machine with Information Gain and Weighted Support Vector Machine. Information gain selects the best subset of features to classify a point and helps remove some of the sensitivity SVM has to a large set of features. In the standard SVM methodology, a weight for the penalty of misclassification is the same for every datapoint, but in weighted SVM the penalty for individual points are set based on the potential for financial loss which helps catch unusual fraudulent cases. One drawback to SVM is that the computation is at least quadratic to the size of the data (Zhang et al., 2020), which makes it hard to train on a large dataset so it is best suited for a balanced dataset using undersampling.

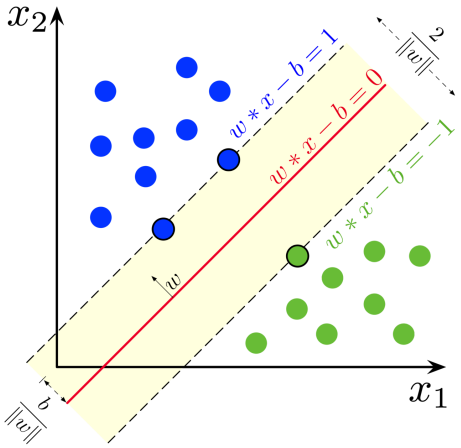

Figure 1. Example plot of Support Vector Machine. *By Larhmam, 2018, image licensed under Creative Commons.*

2.2 Logistic Regression

The logistic regression model is a classification model that predicts the probability of a discrete outcome given an input variable. It is a supervised learning model operating on categorical inputs, in this case whether or not a transaction is an instance of fraud. The regression part represents a regression model where the reliant variables categorize and analyze the relationship between several independent variables (Mishra & Pandey, 2021). The formula for logistic regression is

$$y = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$$

where b is the bias. Logistic regression predicts the probability of an output which has two binary values, true (1) or false (0), by producing logistic curves in this case identifying a transaction as fraudulent or not (Mishra & Pandey, 2021). The model is very efficient, easily tunable, and easy to interpret. Logistic regression can operate without making any assumptions about the distribution of classes within the feature space (Alenzi & Aljehane, 2020), which is key when working with real-world datasets that often exhibit complex patterns. A challenge of using logistic regression to detect fraud is the unbalanced nature of the fraud datasets, since most transactions are normal. The logistic regression algorithm has been employed in conjunction with k-fold machine learning techniques to address the challenge of limited fraud-related data, allowing for cross-validation and improving the accuracy of the model (Mishra & Pandey, 2021). Resampling techniques are also used to address the issue, converting the data into a balanced set of positive and negative samples (Wang & Zhao, 2022). Undersampling removes the training dataset examples which pertain to the majority class to reduce the skew to a 1:1 class distribution (Mqaldi et al., 2021). One such method is NearMiss undersampling, which balances the dataset by randomly eliminating majority class examples. There are three versions of this, NearMiss-1 finds the test data with the smallest average distance to the closest minority class samples, NearMiss-2 chooses the majority samples with the smallest average distance to the minority class's farthest samples and NearMiss-3 is a two-step procedure, preserving the nearest neighbors of each minority sample and choosing majority samples based on the average distance between them and their nearest neighbors (Mqaldi et al., 2021). Logistic regression has a higher average response time when compared to other models such as KNN.

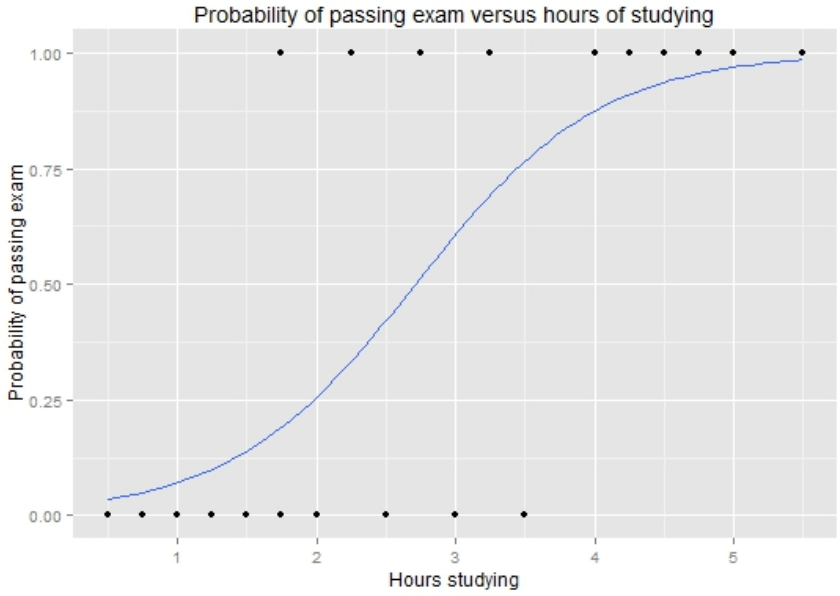

Figure 2. Example logistic regression plot. *By Michaelg2015, 2015, licensed under Creative Commons.*

2.3 K-Nearest Neighbors

K-Nearest Neighbors is a supervised learning algorithm that can be used for both classification and regression. At its core, KNN relies on the principle of similarity. When a new credit card transaction is presented for classification, KNN identifies its k-nearest neighbors in the feature space based on chosen distance metrics (e.g., Euclidean distance). These neighbors act as 'voters' to determine the class of the transaction. If a majority of the nearest

neighbors are labeled as fraudulent, the transaction is classified as fraudulent, and vice versa (Madhumitha & Parthipan, 2023). One of the advantages of KNN is its adaptability to changing fraud patterns. As fraudsters constantly evolve their tactics, KNN can quickly adapt to new trends since it doesn't rely on a fixed model but rather memorizes the data distribution (AlEmad, 2022). This can make it a valuable tool for detecting novel and previously unseen fraud schemes. In the context of credit card fraud detection, KNN works by calculating the distance between a given transaction and its k-nearest neighbors in a feature space. These neighbors are typically chosen based on a distance metric. The better that metric reflects label similarity, the better the classification will be. The most common choice is the Minkowski distance or Euclidean distance. To classify a transaction as either legitimate or fraudulent, KNN looks at the labels of its k-nearest neighbors and assigns the majority class label to the transaction in question. For example, if a transaction has five nearest neighbors, and three of them are known to be fraudulent while two are legitimate, KNN would classify the transaction as fraudulent.

In the context of imbalanced class distributions often encountered in credit card fraud detection, KNN can be fine-tuned to handle this challenge (Makki et al., 2018). Techniques such as oversampling the minority class or assigning different weights to different neighbors can be employed to improve its performance. KNN can be computationally intensive, particularly when dealing with large datasets, as it requires calculating distances between the test point and all training data points. Techniques like dimensionality reduction and efficient data structures (e.g., KD-trees) can help mitigate this issue. While KNN is a viable option for credit card fraud detection, it's essential to consider its limitations, including sensitivity to the choice of the number of neighbors (k) and the distance metric used (Mehbodniya et al., 2021). Furthermore, KNN may not perform as efficiently as Random Forest in handling large datasets or datasets with high dimensionality.

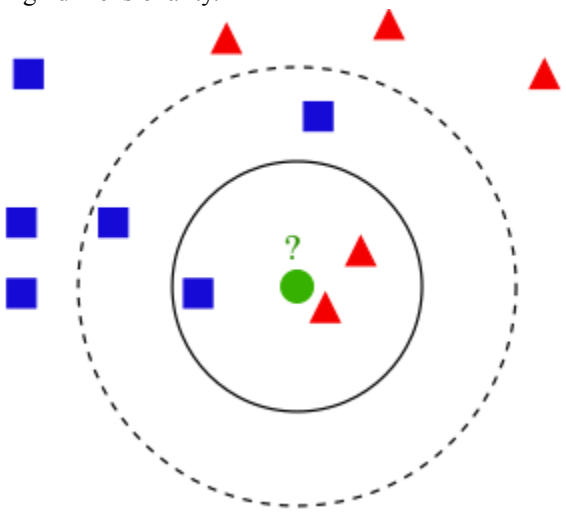

Figure 3. Example plot of K-Nearest Neighbors, with the inner circle representing k=3, and the outer circle representing k=5. *By Antti Anjanki, 2007, image licensed under Creative Commons.*

2.4 Naïve Bayes

Naïve Bayes Classifier is based on Bayes' Theorem for conditional probability, or the probability that given event A occurs, given that event B has already occurred. The theorem is shown below:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

In terms of our dataset, we are looking to find the probability of credit card fraud given a number of factors $X = x_1, x_2, \dots, x_n$. We can rewrite Bayes' Theorem as:

$$P(y|X) = \frac{P(X|y)\,P(y)}{P(X)}$$

where y is the event of credit card fraud. As stated earlier, Naïve Bayes has two major assumptions: it assumes that the features are independent and each have equal contribution to the outcome. Thus, we can rewrite our equation as follows:

$$P(y|x_1, x_2, \dots, x_n) = \frac{P(x_1|y)P(x_2|y)\dots P(x_n|y)\,P(y)}{P(x_1)P(x_2)\dots P(x_n)}$$

$$= \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1)P(x_2)\ldots P(x_n)}$$

As the denominator will remain constant for a given input, we can remove that term:

$$P(y|x_1, x_2, \ldots, x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y)$$

Finally, the classifier model will find the probability of given set of inputs for all possible values of the class variable y and pick the output with maximum probability. This can be expressed mathematically as follows:

$$y = argmax_y P(y)\prod_{i=1}^{n} P(x_i|y)$$

Naïve Bayes Classifier is a popular method today because it is based on probability making it simple and easy to understand. It also does not require as large of a dataset as other methods require, allowing for faster computation. If the assumptions are satisfied and the features are independent, Naïve Bayes is optimal; no other classifier is expected to have a smaller misclassification error rate (Ranganathan et. al, 2019). These assumptions, however, do not always hold. It is very unlikely that the features will be completely independent. Still, the method has demonstrated excellent performance even when this assumption does not hold (Ranganathan et. al, 2019).

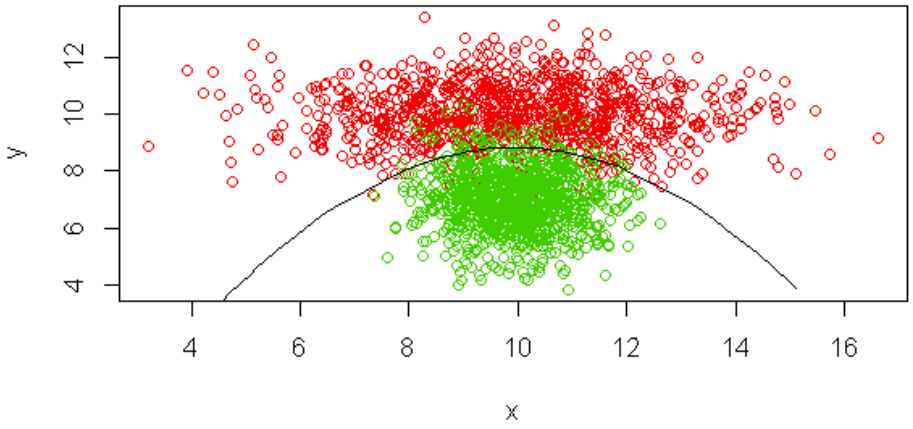

Figure 4. Example plot of a Naive Bayes classifier with the distribution curve. *By Kevin Pei, 2015, image licensed under Creative Commons.*

3. Proposed Solution and Process

An outline of our proposed solution is shown in Figure 5 below. Figure 6 describes the classification of one transaction as either fraudulent or not using our model.

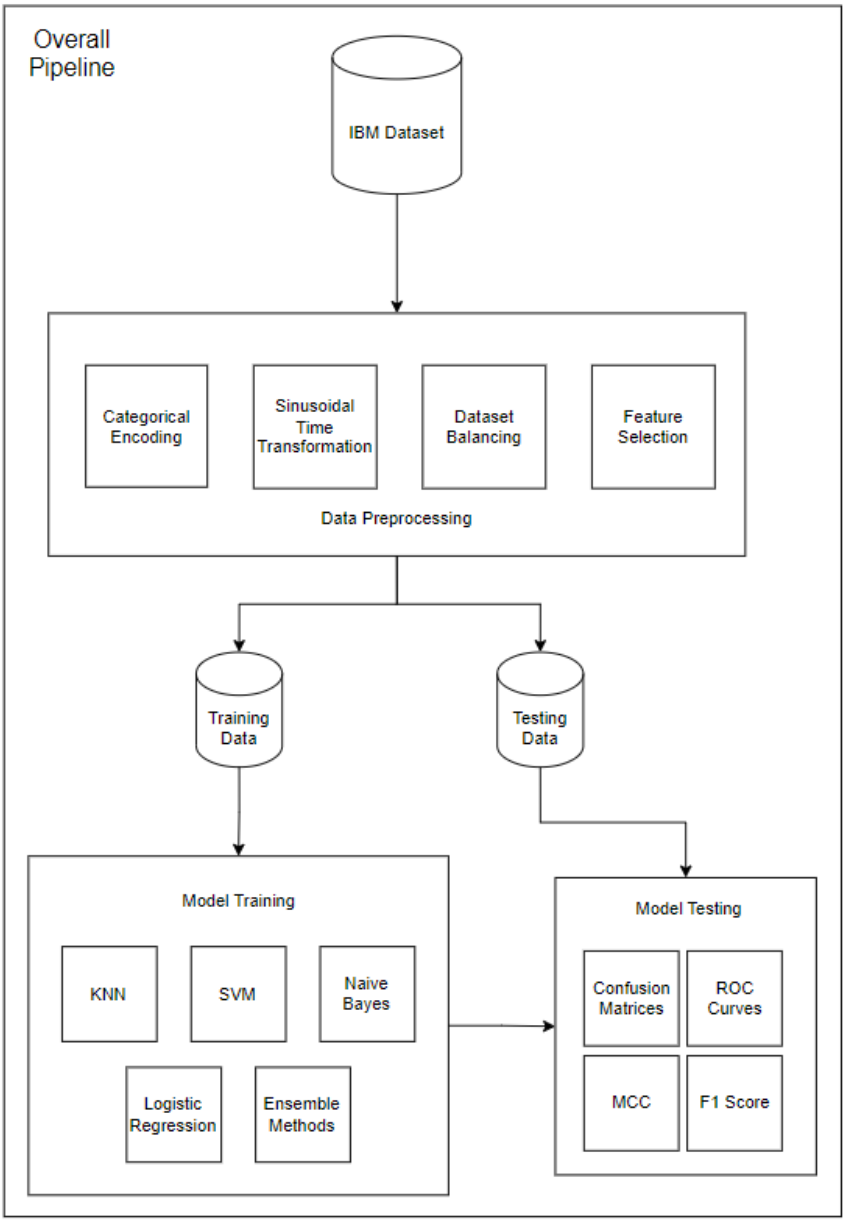

Figure 5. High-level workflow pipeline of our solution.

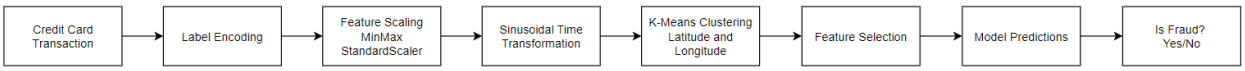

Figure 6. Pipeline of classifying a single transaction. The transaction passes through the data preprocessing pipeline described below, then our trained and tuned models will make a prediction and classify the transaction as fraudulent or not fraudulent.

Our solution consisted of preparing our data, training four models: K-Nearest Neighbors, Random Forest, Naive Bayes, and Logistic Regression, testing and validating these four models, and then comparing various performance metrics between the models. Our data preparation consisted of balancing our dataset, deriving additional features from our dataset, encoding our categorical features, transforming our time features, and selecting relevant features. Our dataset was imbalanced, with only around 30,000 fraudulent cases, 0.12% of the overall dataset. An imbalanced

dataset can result in poor sensitivity, as a classifier could simply classify every instance as not fraud and have a 99.88% accuracy, but not classify any of the fraudulent cases correctly. In the context of our problem, we want to minimize the number of fraudulent cases that get classified as non-fraudulent, maximizing the sensitivity. To address the imbalance, we utilized the Near Miss 2 undersampling technique. This method selects the instances from the majority class that have the closest average distance to the three furthest instances from the minority class. We implemented this method using the imblearn.under_sampling library in Python. After our balancing, our dataset had 59,515 rows, and a 50% split for fraudulent to non-fraudulent cases. An outline of our data preprocessing is shown below in Figure 7.

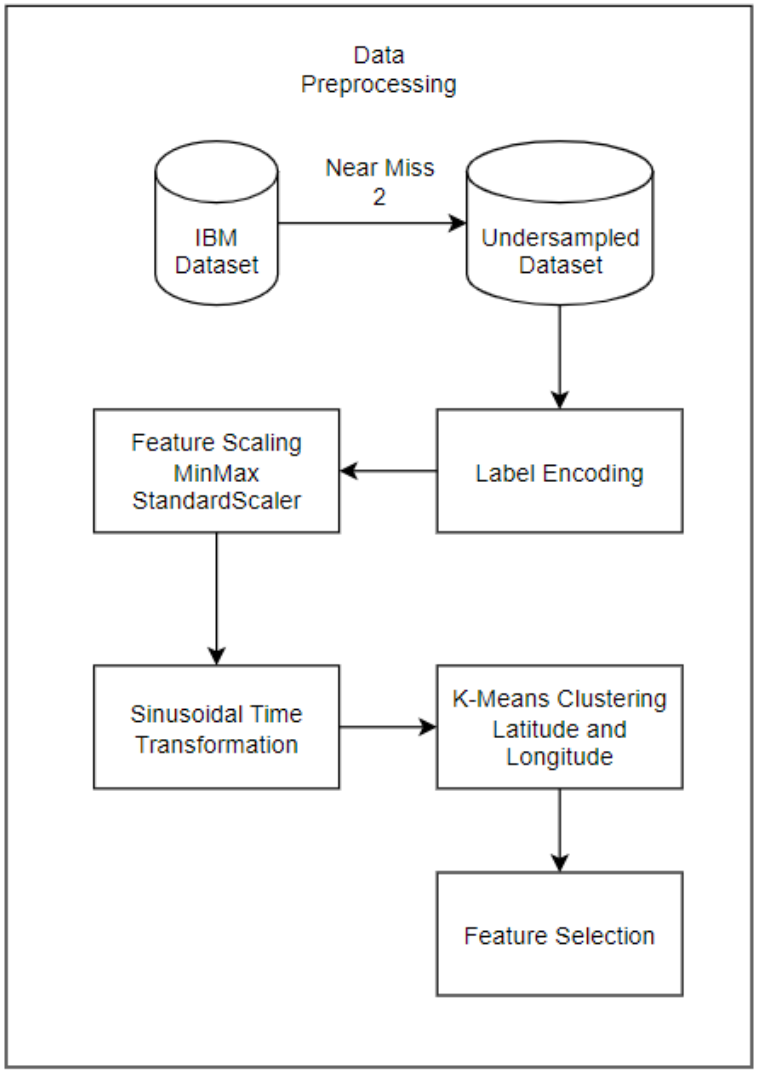

Figure 7. Pipeline of our data preprocessing steps. Our raw dataset is undersampled using Near Miss 2 to balance our target classes. Our categorical features are then label encoded, our numerical features are scaled using MinMax scaling and Standard Scaling. Our time features are transformed into sinusoidal values, and latitude and longitude are clustered using the KMeans algorithm.

To encode our time features, we used a sinusoidal transformation, using sine and cosine to transform months, days, and hours to a value between -1 and 1. This transformation captures the cyclical nature of these time periods, to preserve the relationship between months, days, and hours. To perform this transformation, we used the CyclicalFeatures class of the feature_engine.creation Python module. This is done in two steps, converting the periods into radian angles from 0 to 2π, and then finding the sine and cosine of these radian angles. For example, for

the 24 hour range each hour is (x/24)*2π. Performing this transformation resulted in 'Month_sin', 'Month_cos', 'Day_of_Week_sin', 'Day_of_Week_cos'. 'Day_sin', 'Day_cos', 'Hour_sin', 'Hour_cos'.

To encode the merchant category code (MCC) feature, we binned the values by the industry the MCC belongs to using the mapping seen in Figure 8.

```
def mcc_mapping(mcc):
  if mcc < 700:
    return 'Reserved'
  elif mcc < 1000:
    return 'Agricultural services'
  elif mcc < 1500:
    return 'Reserved'
  elif mcc < 3000:
    return 'Contracted services'
  elif mcc < 4000:
    return 'Reserved for private use'
  elif mcc < 4800:
    return 'Transportation'
  elif mcc < 5000:
    return 'Utilities'
  elif mcc < 5500:
    return 'Retail outlets'
  elif mcc < 5600:
    return 'Automobiles and vehicles'
  elif mcc < 5700:
    return 'Clothing outlets'
  elif mcc < 6000:
    return 'Miscellaneous outlets'
  elif mcc < 7300:
    return 'Service providers'
  elif mcc < 7530:
    return 'Business services'
  elif mcc < 7800:
    return 'Repair services'
  elif mcc < 8000:
    return 'Amusement and entertainment'
  elif mcc < 9000:
    return 'Professional services and membership organizations'
  elif mcc < 9200:
    return 'Reserved for ISO use'
  elif mcc < 9403:
    return 'Government services'
  elif mcc < 10000:
    return 'Reserved'
  else:
    return 'UNK'
```

Figure 8. Encoding of MCC codes by industry. Encodings taken from www.classification.codes/classifications/industry/mcc/.

In addition to the 45 features present in the dataset, we generated additional features using the dataset. The features we engineered were: the number of months since the PIN was changed, the number of months since the account was opened, the ratio of total debt to credit limit for a user, the ratio of a user's yearly income to total debt, the ratio of a user's yearly income to credit limit, the ratio of amount of the transaction to overall credit limit (credit limit utilization), number of cards a user has relative to the average, number of cards a user has relative to the median, a binned user age feature, with bins for 20-34, 35-49, 50-65, and over 66 years old, whether or not a user made a purchase in the last 3, 6, 9, 12, 15, 18, and 24 months, the amount a user has spent in the last 3, 6, 9, 12, 15, 18, and

24 months, the average amount spent in transactions with a merchant in each state and online, the average amount spent in transactions with a certain merchant category code (MCC), and combining the card brand and type. We used a FICO score breakdown from Experian to create a binned FICO score feature, with bins for poor, fair, good, very good, excellent (Akin, 2020).

After our feature generation, we had 66 total features. Before feature selection, we encoded all of the categorical features, as seen in Table 1. For latitude and longitude, we used KMeans clustering, from the sklearn.clustering Python library to cluster the locations of users into four clusters. We label encoded twelve features. We used the StandardScaler class from the sklearn.preprocessing Python library to standardize 28 features. We also applied the MinMaxScaler class from sklearn.preprocessing to standardize 35 features. Standard Scaler normalizes data points by subtracting the mean of the feature from the original value and then dividing by the standard deviation. We chose this normalization method for features that had a roughly Gaussian distribution. MinMax Scaler normalizes numerical values by first subtracting the minimum value from the original value, then dividing by the range of the feature. That value is then multiplied by the desired range, and the desired minimum is then added to it. We used 0 and 1 as the desired minimum and maximum values. We chose this normalization technique for the features that did not follow a bell curve or Gaussian distribution. The details of the categorical encoding and transformations are shown below.

Label Encoded	**StandardScaler**	**MinMaxScaler**
Card Brand	Cards Issued	Credit Limit
Card Type	Per Capita Income - Zipcode	Year PIN last Changed
Has Chip	Yearly Income - Person	Current Age
Gender	Num Credit Cards	Retirement Age
State	Amount	Birth Year
Use Chip	Year	Latitude
Merchant State	#Pin Changed by User	Longitude
Errors	Avg_MCC	Total Deb
State_US_Region	Avg_State	FICO Score
Merchant Country	Day_of_Week	Month
MCC	Month_sin	Day
Card Brand Type	Month_cos	MCC
	Day_of_Week_sin	#Transaction by Use
	Day_of_Week_cos	Hour
	User_Active_Past_3_Months	Day_sin
	User_Active_Past_6_Months	Day_co
	User_Active_Past_9_Months	Hour_sin
	User_Active_Past_12_Months	Hour_cos
	User_Active_Past_15_Months	User_Amount_Spent_Past_3_Months
	User_Active_Past_18_Months	User_Amount_Spent_Past_6_Months
	User_Active_Past_21_Months	User_Amount_Spent_Past_9_Months
	User_Active_Past_24_Months	User_Amount_Spent_Past_12_Months
	FICO_Score_Binned_Ordinal	User_Amount_Spent_Past_15_Months
	User_Age_Binned_Ordinal	User_Amount_Spent_Past_18_Months
	Debt_Credit_Limit_Ratio	User_Amount_Spent_Past_21_Months
	Yearly_Income_To_Total_Debt_Ratio	User_Amount_Spent_Past_24_Months
	Yearly_Income_To_Credit_Limit_Ratio	Expiration Year
	Num_Of_Cards_Relative_To_Median	Account_Open_Year

		PIN last Changed Account Open Age Years in Card Expiration Retirement to Current Credit_Limit_Utilization Years PIN Change After Account Open Months_Since_Account_Open

Table 1. Categorical encodings and transformations of features.

After preprocessing our dataset, we split our data into training and testing sets using the train_test_split function from the sklearn.model_selection library. We used an 80/20 split, or 80% in the training set and 20% in the testing set. We stratified our split using our target variable, Is Fraud?. We did this in order to ensure we had a balanced distribution of our target variable in the two datasets.

Using these encodings, we created every unique combination of features, and ran a Random Forest model on each combination using the RandomForestClassifier from the sklearn.ensemble Python library. We then used a two-tier ensemble based feature selector, consisting of three main feature selection methods: wrapper, embedded, and filter methods (Markus et al., 2022). For each of these types of feature selection, we used a set of ML algorithms to select features for our models. Embedded methods perform feature selection during the training of the regressor (Markus et al., 2022). The three embedded methods we used were Lasso, Ridge, and Elastic-Net, adding a regularization term to the objective function to select the best features. Lasso works well with high-dimensional datasets such as the credit card fraud dataset, setting the coefficients of redundant features to zero, while Ridge is best for dealing with multicollinearity among the features and reduces the impact of these features in the model prediction. ElasticNet is a combination of Lasso and Ridge, using a ratio of both regularization terms to select the best features.

Filter methods select features based on their scores in various statistical tests for their correlation with the outcome variable. The filter methods we used are mutual information, ANOVA f-test, and chi-squared. Mutual Information estimates the mutual information for fixed categories like in a classification problem (Guhanesvar, 2021). This test, unlike the other two used, can be used for both numerical and categorical features. ANOVA, or analysis of variance, is a parametric statistical hypothesis test for determining whether the means from two or more samples of data come from the same distribution or not (Brownlee, 2020). An f-test calculates the ratio between variance values (Brownlee, 2020). This test is mainly used when you have numerical features and a categorical output. Finally, the chi-squared test is used to test the independence of two events (Gajawada, 2019). This test is used when you have categorical input and output features. For each of these tests, a higher value signifies a stronger relationship with the target variable. In feature selection, we want to select the features which are highly dependent on the response (Gajawada, 2019).

Wrapper methods use a greedy search approach by evaluating all the possible combinations of features against the evaluation criterion (Brownlee, 2020). We used recursive feature elimination, a wrapper method that uses a base classifier to select a subset of features that are important (Brownlee, 2020). We used a Random Forest classifier as the base model and selected a subset of 20 features.

After performing each of these feature selection techniques, we compared the results and kept the features that were chosen by all the methods. The resulting list of features can be seen in Table 2 below.

Feature Name	# of Models Selecting Feature
User Amount Spent Past 3 Months	6
Per Capita Income - Zip Code	5
Yearly Income - Person	5
Avg_State	5
User Amount Spent Past 6 Months	5
User Amount Spent Past 9 Months	5
User Amount Spent Past 15 Months	5
User Amount Spent Past 18 Months	5
User Amount Spent Past 24 Months	5
User Amount Spent Past 12 Months	4
User Amount Spent Past 21 Months	4
Current Age	4
Total Debt	4
Hour_cos	4

Table 2. 14 Selected Features from Two-Tier Ensemble Based Feature Selection.

With our subset of features, we trained our four models on the training data, as seen in Figure 9. We used the GaussianNB class from the sklearn.naive_bayes Python library, the KNeighborsClassifier class from the sklearn.neighbors library, the LogisticRegression class from the sklearn.linear_model library, and the RandomForestClassifier model from the sklearn.ensemble library.

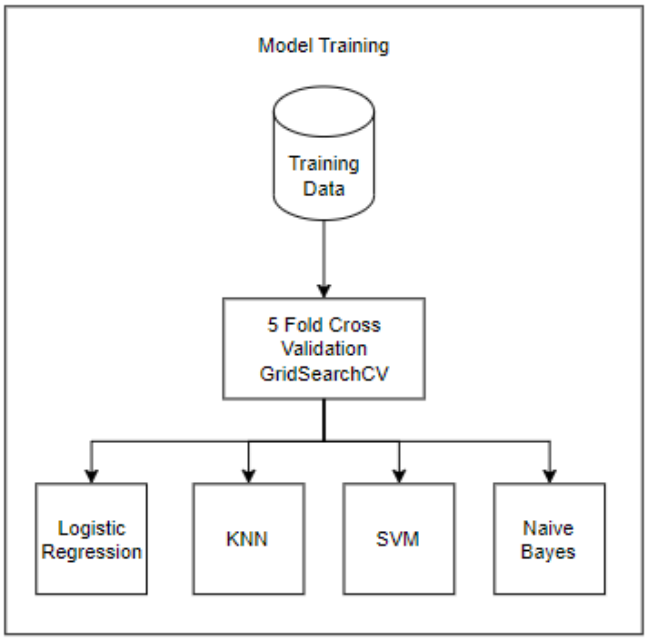

Figure 9. The process of our model training.
Our four models are trained and tuned using K-Fold cross-validation with five folds.

```
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
y_pred_nb = nb_classifier.predict(X_test)
accuracy_nb = accuracy_score(y_test, y_pred_nb)

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
accuracy_knn = accuracy_score(y_test, y_pred_knn)

lr_classifier = LogisticRegression(random_state=123, solver='lbfgs', C=0.1, max_iter=1000)
lr_classifier.fit(X_train, y_train)
y_pred_lr = lr_classifier.predict(X_test)
accuracy_lr = accuracy_score(y_test, y_pred_lr)

svc_classifier = SVC(random_state=123)
svc_classifier.fit(X_train, y_train)
y_pred_svc = svc_classifier.predict(X_test)
accuracy_svc = accuracy_score(y_test, y_pred_svc)
```

Figure 10. Code implementation for training and testing our four baseline models.

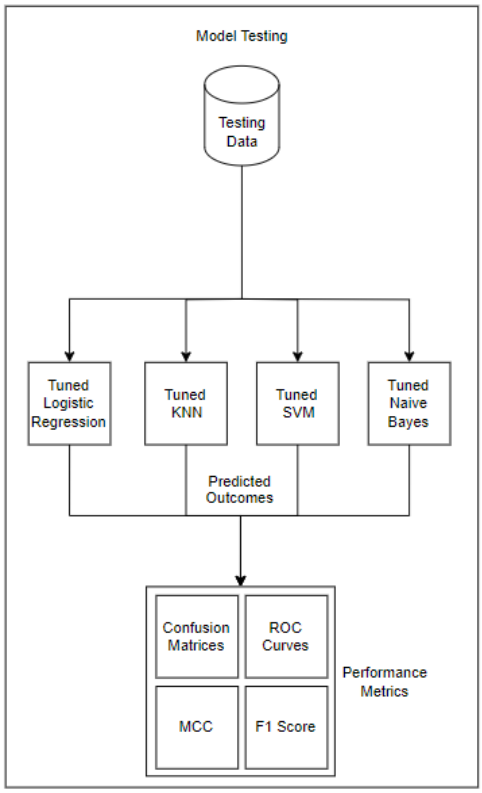

Figure 11. The pipeline of our model testing. Our tuned models are tested on our testing set, and we represent our results through confusion matrices, ROC curves. In addition to these, we measured the Matthews Correlation Coefficient and the F1 score of the models.

We tested our models using the default parameters to get a baseline performance. After this, we performed hyperparameter tuning on the four models to increase performance.

Finally we tried ensemble methods to further increase the performance of the solution.We tried bagging, boosting, stacking, and a voting classifier. Bagging is an ensemble classifier that fits base classifiers on random subsets of the original dataset and then aggregate their individual predictions to form a final prediction. Bagging generates several training models by providing each model with overlapped training data subsets for modeling. We created bags for each of our tuned models and tested their performance.

Boosting is an ensemble method that trains models sequentially. The first model is trained on the dataset, and its incorrect predictions are used to train the next model. By doing this, subsequent models can improve on the errors of previous models. The results are aggregated at each step using a weighted average that gives more weight to the model with the highest predictive power.

Stacking is an ensemble method that uses the output of one model as an input feature for the next model in the stack. Finally, the last model in the stack produces a prediction. We tried this method with each of our four models as the final estimator, as well as trying to use a random forest model as the final estimator.

Voting Classifier involves combining the predictions of multiple models to achieve improved overall performance. The process involves aggregating the findings of each classifier passed into Voting Classifier and predicting the output class based on the highest majority of voting. In hard voting, the predicted output class is a class with the highest majority of votes, or the class which has the highest probability of being predicted by each of the classifiers.

4. Experimental Results and Discussion

To get a baseline performance measurement for our four models, we measured the accuracy, precision, sensitivity, Matthews Correlation Coefficient, F1 Score, and ROC curve for each of our models with the default parameters. The results of these experiments can be seen in Table 3.

	Accuracy	**Precision**	**Sensitivity**	**MCC**	**F1 Score**
Naive Bayes	82.35%	74.62%	95.70%	0.68	0.84
KNN	92.75%	89.98%	95.50%	0.86	0.93
Logistic Regression	83.86%	76.38%	95.97%	0.70	0.85
Support Vector Machine	83.07%	74.75%	97.63%	0.69	0.85

Table 3. Baseline performance metrics for our four models.

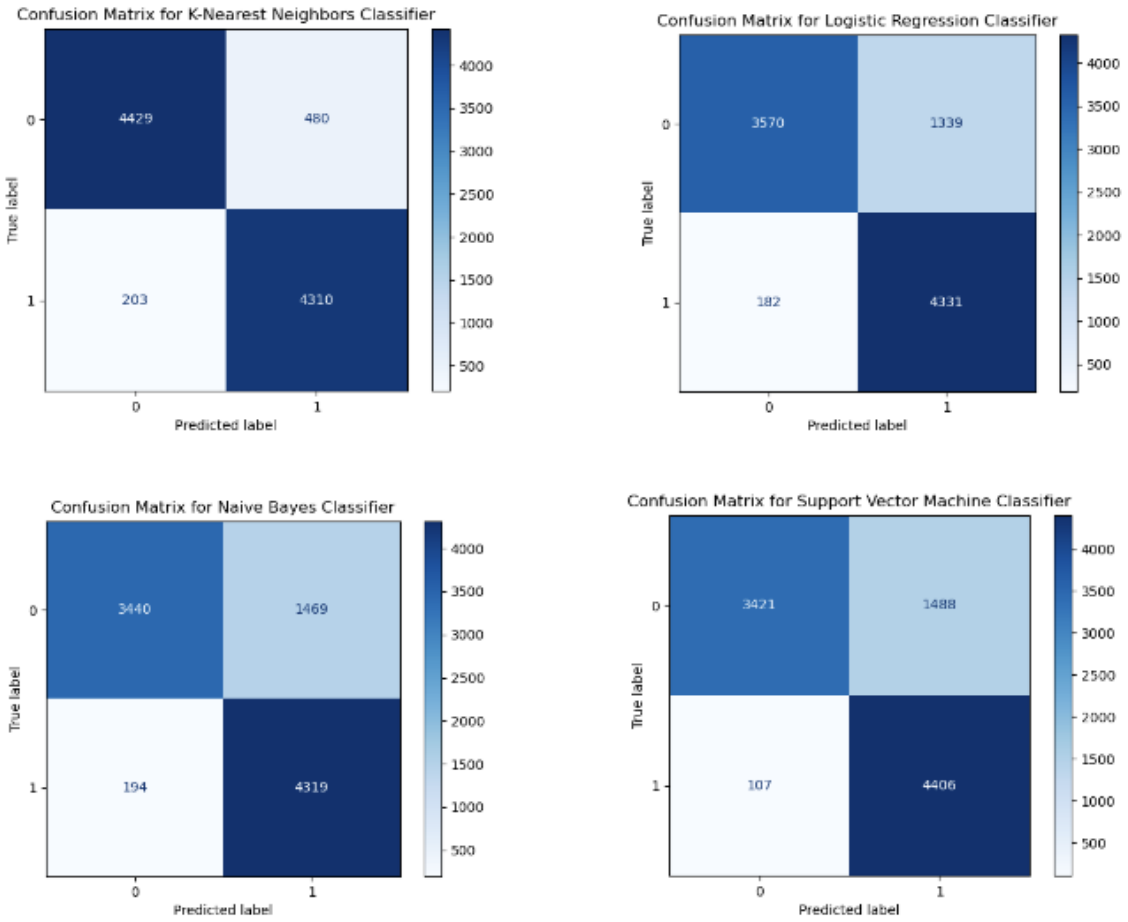

Figure 12. Confusion matrices for the four models.

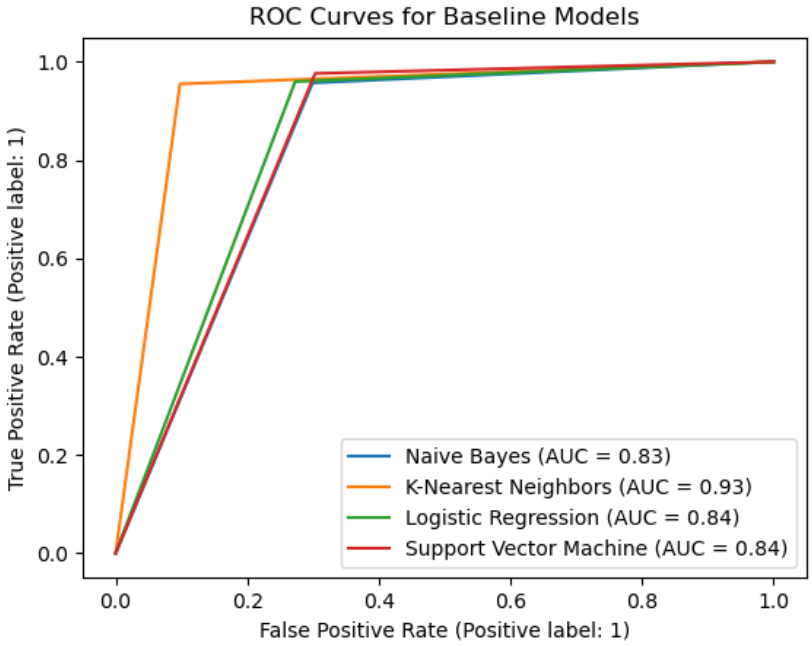

Figure 13. ROC Curves for the four baseline models.

To measure the models' accuracy, precision, and sensitivity, we used a confusion matrix, implemented with the confusion_matrix function from the sklearn.metrics Python library. After creating a confusion matrix for each of the four models, as seen in Figure 12, we measured the accuracy using the accuracy_score function from the sklearn.metrics library. We calculated the precision as the number of true positives divided by the number of true positives plus false positives. This gives a measure of the percentage of positive predictions that were correct. To calculate sensitivity, we divided the total number of true positives by the number of true positives plus false negatives. This gave us a measurement of the percentage of positive cases the models predicted correctly.

To measure the Matthews Correlation Coefficient, we used the matthews_corrcoef function from the sklearn.metrics library. This measured the quality of our models, with a coefficient of one representing a perfect classification, and zero being a random prediction quality. To measure F1 score, we used the f1_score function from the sklearn.metrics library. F1 Score is equal to $(2 \star precision \star sensitivity) / (precision + sensitivity)$. One is the best score it can achieve, and zero is the worst.

From the preliminary baseline metrics, the Support Vector Machine classifier has the highest sensitivity, with 97.63%, but all four models perform over 95%. In all four models, we see a higher sensitivity than precision, with almost 20 percent differences for the Logistic Regression, Naive Bayes, and Support Vector machine models. The K-Nearest Neighbors model has the highest MCC, F1 Score, and AUC value. While the SVM has the highest sensitivity, the KNN model has a very high sensitivity as well, and significantly outperforms the other three models in accuracy, precision, MCC, F1 score, and AUC value, so the KNN model appears to be the highest performing model before any hyperparameter tuning.

For hyperparameter tuning, we used the GridSearchCV class in Sklearn, with 5-fold cross validation, and the Matthews Correlation Coefficient as the performance metric. For the Naive Bayes model, we tuned the var_smoothing parameter. The best value we found was 6e-10, but did not see an increase in performance over the baseline. For the K-Nearest Neighbors model, we tuned the number of neighbors, the p value, and the algorithm used. We found parameters of one neighbor, a p value of 2, and the 'auto' algorithm. We saw a small performance

increase with these parameters, with accuracy increasing from 92.75% to 94.2%, the MCC increasing from 0.86 to 0.88, and the F1 score increasing from 0.93 to 0.94. For the Logistic Regression mode, we tuned the penalty, C value, and the solver. We found the 'newton-cg' solver and no penalty value to be the best set of parameters. With these parameters, accuracy increased from 83.86% to 84.72%, the MCC increased from 0.7 to 0.71, and the F1 score increased from 0.85 to 0.86. For the Support Vector Machine, we tuned the C value, the kernel, and the gamma value. We found that a C value of 1.5, a gamma value of 'scale', and the 'rbf' kernel were the best set of parameters. We saw almost no increase in accuracy, from 83.07% to 83.21%, the MCC increased from 0.69 to 0.7, and no increase in F1 score.

	Accuracy	**Precision**	**Sensitivity**	**MCC**	**F1 Score**
Naive Bayes	82.35%	74.62%	95.70%	0.68	0.84
KNN	94.18%	93.07%	94.93%	0.88	0.94
Logistic Regression	84.72%	77.95%	94.95%	0.71	0.86
Support Vector Machine	83.21%	74.92%	97.63%	0.70	0.85

Table 4. Performance metrics for our four models after hyperparameter tuning.

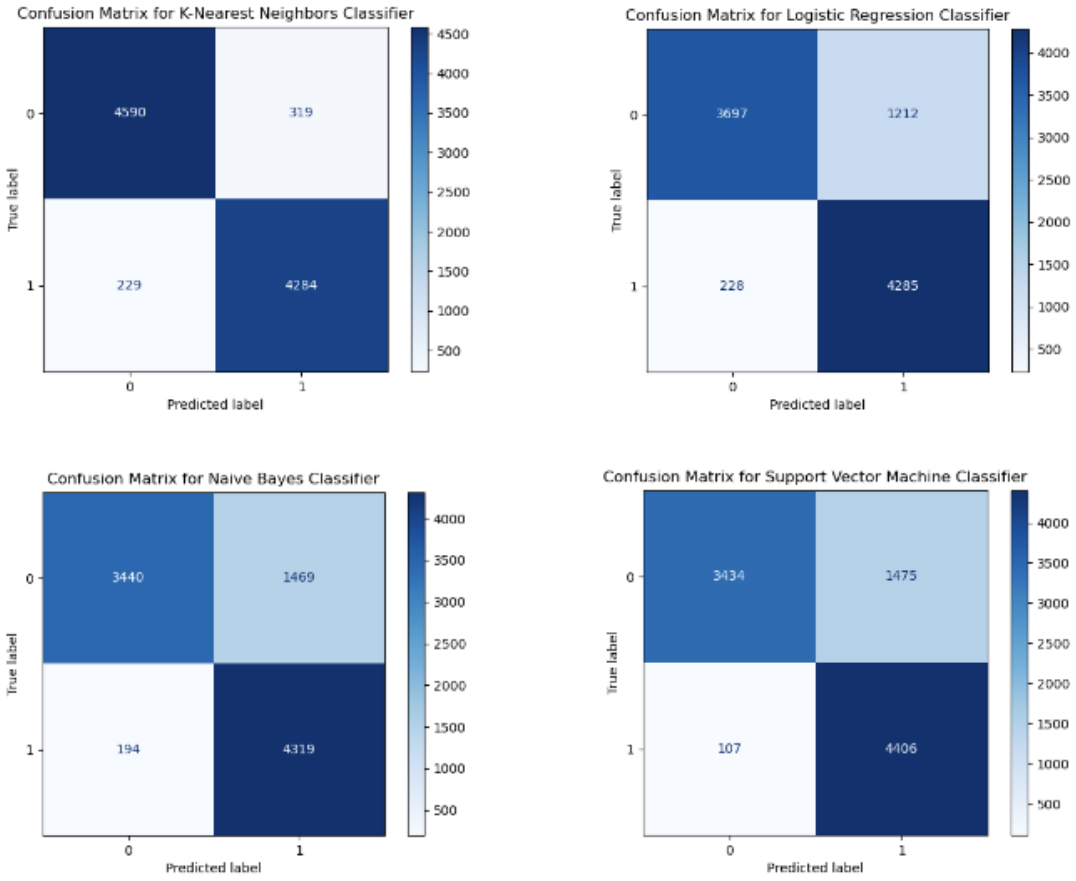

Figure 14. Confusion matrices for the four models after hyperparameter tuning.

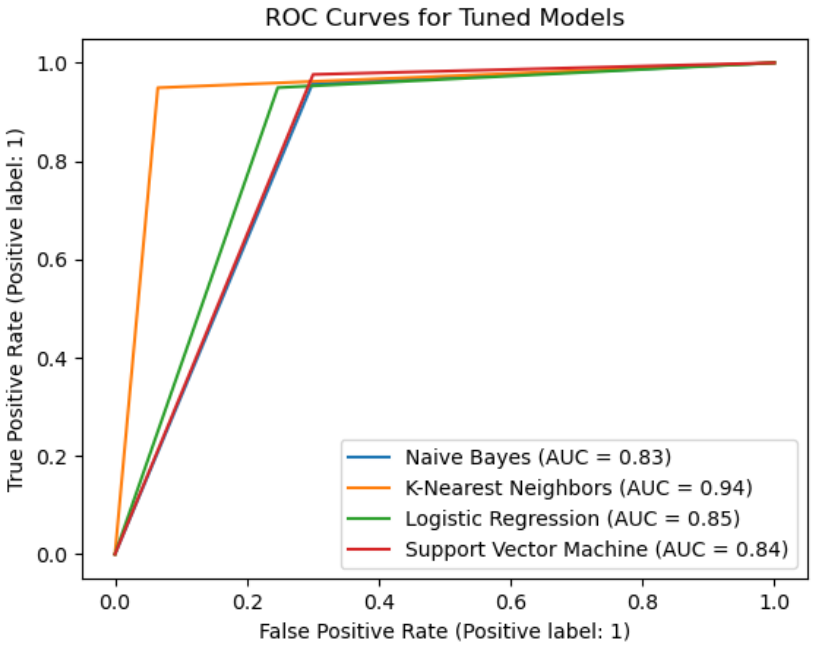

Figure 15. ROC Curves for the tuned SOTA models.

After hyperparameter tuning, we wanted to see if we could improve the performance even more by using an ensemble of models. We tried four different ensemble methods: bagging, boosting, stacking, and a voting classifier.

Final Estimator	**Accuracy**	**Precision**	**Sensitivity**	**MCC**	**F1 Score**
Logistic Regression	84.76%	77.98%	95.01%	0.71	0.86
K-Nearest Neighbors	94.02%	93.05%	94.59%	0.88	0.94
Naive Bayes	82.37%	74.65%	95.68%	0.68	0.84
SVM	83.22%	74.92%	97.67%	0.70	0.85

Table 5. Performance metrics for our four bagging ensemble classifiers, with each of the models being used as the final estimator.

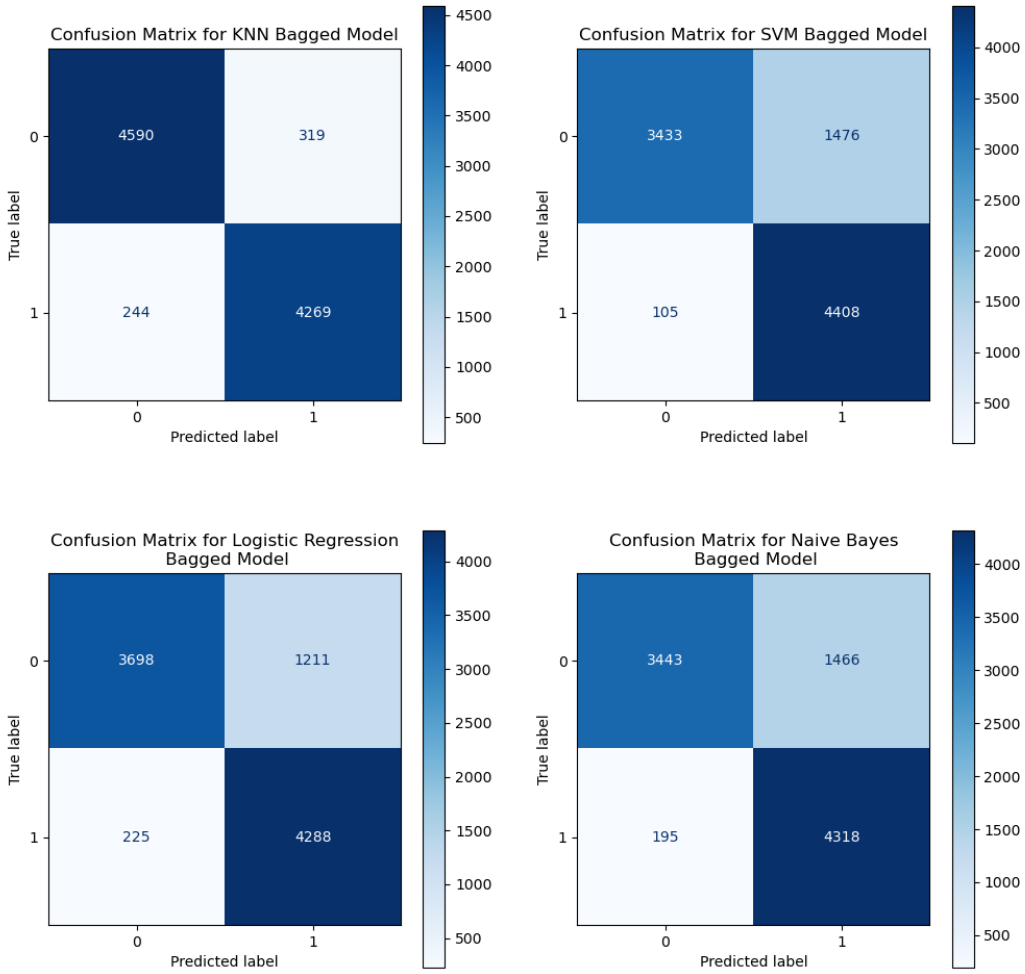

Figure 16. Confusion matrices for the four bagged models with each of the four SOTA models as the base estimator.

Bagging with each of the four tuned SOTA models as the base estimator did not offer any improvements. The performance of the bagged models were all the same as the base estimators at detecting instances of fraud.

Base Estimator	**Accuracy**	**Precision**	**Sensitivity**	**MCC**	**F1 Score**
Logistic Regression	83.76%	77.14%	93.93%	0.69	0.85
Naive Bayes	78.92%	80.51%	73.88%	0.58	0.77

Table 6. Performance metrics for our two boosted ensemble classifiers, with Logistic Regression and Naive Bayes being used as the final estimator.

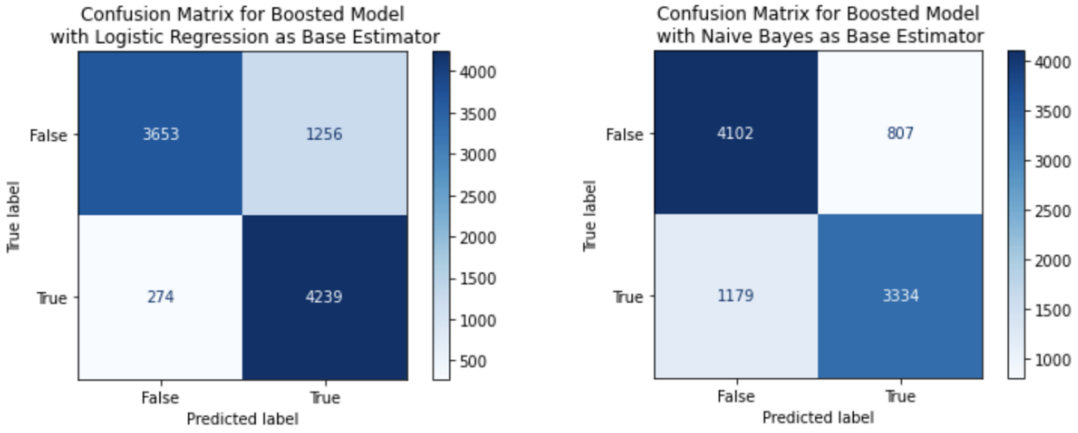

Figure 17. Confusion matrices for two boosted models with Logistic Regression and Naive Bayes as the base estimator.

Boosting with each of our four SOTA models as the base estimator was only successful for two out of our four models. Also, for those that were successful, logistic regression and Naive Bayes, we saw a decrease in our evaluation metrics.

Final Estimator	**Accuracy**	**Precision**	**Sensitivity**	**MCC**	**F1 Score**
Logistic Regression	93.92%	93.26%	94.11%	0.88	0.94
K-Nearest Neighbors	89.91%	89.28%	89.78%	0.80	0.89
Naive Bayes	93.18%	93.64%	92.00%	0.86	0.93
SVM	94.18%	93.07%	94.93%	0.88	0.94

Table 7. Performance metrics for our four stacking ensemble models, with each of the models being used as the final estimator.

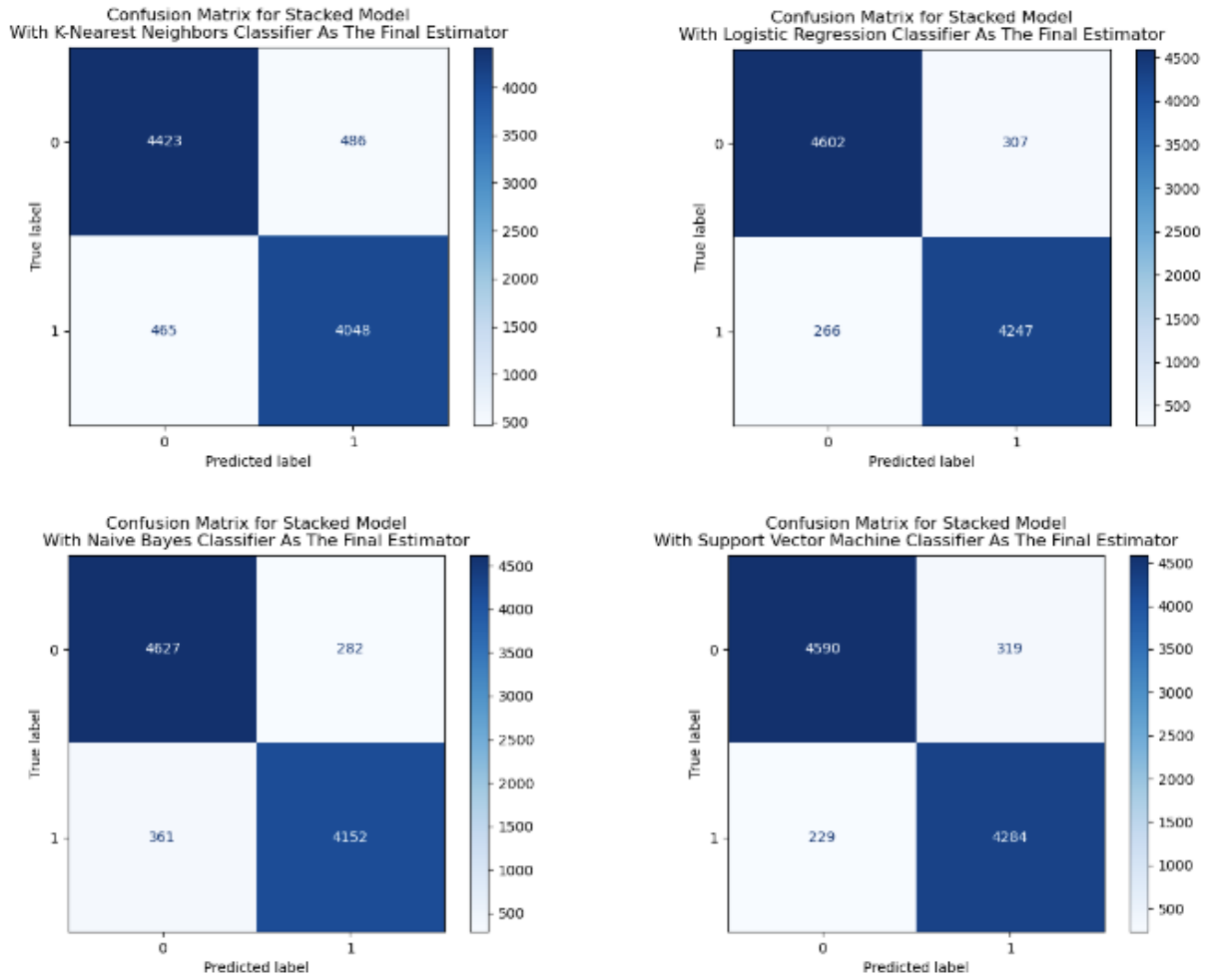

Figure 18. Confusion matrices for the four stacked models with each of the four SOTA models as the final estimator.

We saw increases in performance from stacking in three of the four models, with the largest increase coming from the Naive Bayes and the SVM models, with both seeing an increase of 0.18 in the MCC. The Logistic Regression and SVM stacked models both achieved an F1 score of 0.94 making them the highest performing stacked models, and matching the highest performance overall.

We also tried using a combination of both boosting and stacking. The combination of these two ensemble models showed great results, which can be seen below.

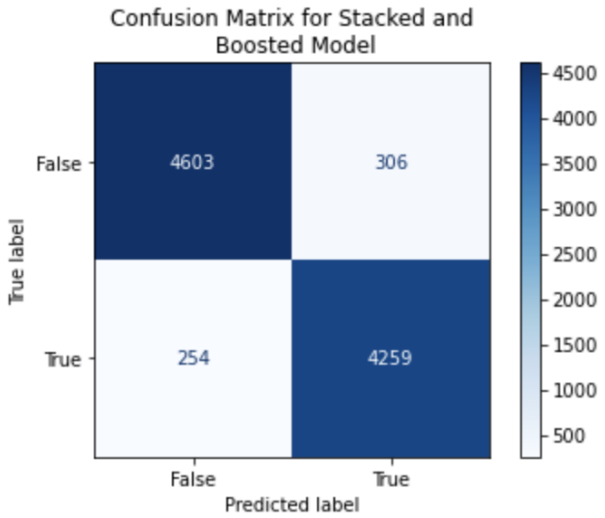

Figure 19. Confusion matrix for boosted and stacked model.

Accuracy	**Precision**	**Sensitivity**	**MCC**	**F1 Score**
94.06%	93.30%	94.37%	0.88	0.94

Table 8. Performance metrics for boosted and stacked model.

Finally, we tried a Voting Classifier that involves combining the predictions of multiple models to achieve improved overall performance. The goal was to generate a single model to enhance the accuracy and robustness by leveraging the diversity and strengths of multiple individual models. The trained models are evaluated on a test dataset, and their accuracies are computed. We identify instances of incorrect predictions for further analysis. This step is crucial in understanding the strengths and weaknesses of each model as it provides insights into areas for improvement and helps gauge the model's reliability. Ensembles are formed by considering all conceivable combinations of the individual models. The VotingClassifier from the scikit-learn library facilitates the implementation of hard voting ensembles. This strategic amalgamation aims to harness the diversity inherent in individual models, fostering an environment where each model contributes distinct insights, collectively resulting in improved overall predictive accuracy. The evaluation spans ensembles with two, three, and four models combined. This meticulous analysis provides nuanced insights into the effectiveness of different ensemble configurations, guiding the selection of the most potent combinations. The best-performing ensemble was on par with the performance shown by KNN model, which can be seen below. The insights gained from this process contribute to a more informed and confident deployment of predictive models in real-world scenarios, where adaptability and resilience are paramount.

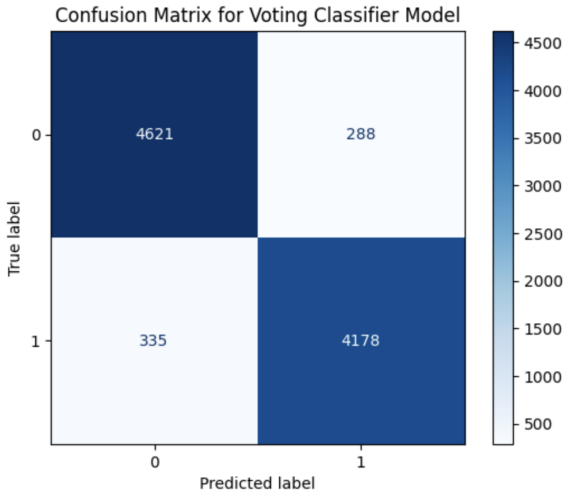

Figure 20. Confusion matrix for Voting Classifier model.

Accuracy	**Precision**	**Sensitivity**	**MCC**	**F1 Score**
93.39%	93.55%	92.58%	0.87	0.93

Table 9. Performance metrics for Voting Classifier model.

5. Lessons Learned

The methods we chose required our variables to be numerical; however, many of our variables were categorical. This meant that we had to encode them as numbers before running our models. We explored multiple different techniques to encode these variables. One of the most common techniques is label encoding. Label encoding labels each of the classes from 0 to $n - 1$ where n is the total number of classes. To do this we can use the LabelEncoder package from the sklearn.preprocessing Python library. One issue with this type of encoding is that it implies an order on the data, and most of our data is not ordinal. When our data is not ordinal, a popular technique we can use is called one-hot encoding. This method creates $n - 1$ columns where each column is one of the classes of the categorical variable. The column contains a 1 if the instance is of that class, and 0 otherwise. We can use the sklearn.preprocessing library to implement this method as well with the OneHotEncoder package. This method is more difficult to interpret since there are now multiple columns for one feature. Thus, we also explored frequency encoding and mean encoding. Frequency encoding is simply replacing the categorical variables with their count. Mean encoding replaces the category with the mean of the target variable for that category. The type of encoding depends on the type of variable. Therefore, we played around with different encodings for different variables in order to determine which ones worked best for our models.

In order to use time variables as predictors for our machine learning models, we learned to use a new method of encoding, cyclical encoding. This encoding has the property that two similar hours will be nearby in the feature space (Mahajan et al., 2021). For example, with the hour of day, the standard representation is an integer ranging from 0 to 23. The problem with this representation is that while midnight (0:00) and 23:00 are only 1 hour apart, the difference between their encodings, 0 and 23 is large. Using the cyclical sine and cosine transformation, the relationship between these hours is maintained as the Euclidean distance between the encoded values for 0 and 23 is the same as 0 to 1 (Figure 21). Additionally, this captures the continuous nature of these values. Previous research

has shown that sin/cos encoding performs as well as other methods of encoding on real world datasets in regression models, and performs better in classification models (Mahajan et al., 2021). We used cyclical encoding on the month, day, and hour using Feature Engine's CyclicalFeatures module. This is achieved by calculating sin(2πx) and cos(2πx) for each of the values of the input variable. This produced 6 columns, month_cos, month_sin, day_cos, day_sin, hour_cos, and hour_sin.

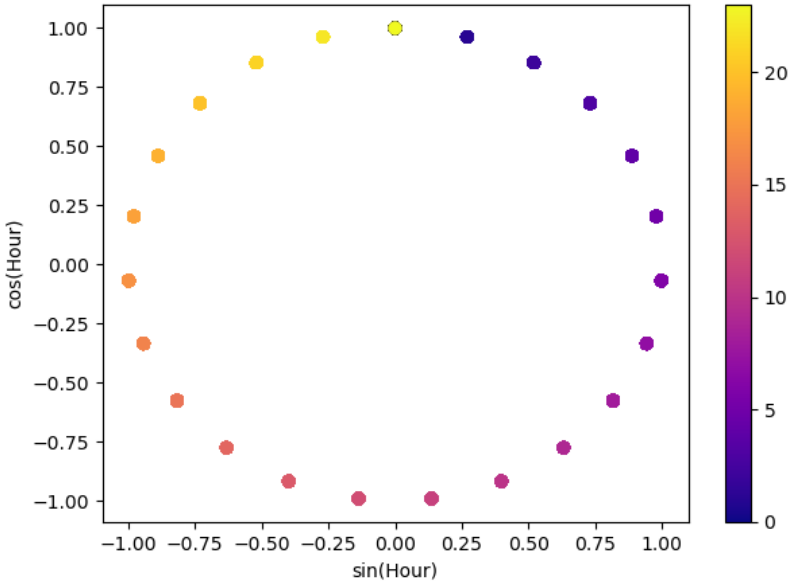

Figure 21. Sine and Cosine encoding of hour, with each point representing one hour of the day.

As we mentioned previously, one challenge we faced was an imbalanced dataset. To deal with this, there are a variety of both undersampling and oversampling techniques. Since our dataset is so large, we decided to use an undersampling technique to reduce the number of instances in our majority class, instead of creating new instances of the minority class. One undersampling technique that we tried is called Condensed Nearest Neighbors (CNN). This method attempts to create a subset of the original data, called a minimal consistent set, that results in no loss to the model performance. It is created by choosing samples to add to the 'store' only if they cannot be correctly classified by the current contents of the store. One issue with this method is that it reduces both the majority and the minority class. Since we have very few instances of fraud we want to keep them all. Therefore, we decided to try another undersampling technique. The next technique we tried is called Near Miss. This technique selects samples based on the distance of the majority class examples to the minority class examples. There are three different versions of it. We chose to use version 2 which selects examples from the majority class that have the smallest average distance to the three furthest examples from the minority class (Brownlee, 2021). This method worked well for our data, and we successfully reduced the majority class to the same size as the minority class.

6. Conclusions and Future Work

This project has provided valuable insights and demonstrated the effectiveness of various machine learning models in addressing the challenge of identifying fraudulent transactions. The conclusions drawn from the current work pave the way for future endeavors in enhancing the model's capabilities and addressing potential areas of improvement.

We found that out of the four state-of-the-art classification methods we tried, K-Nearest Neighbors performs the best. We also had varying results using ensemble methods. Stacking methods seemed to improve model performance a bit, but not much. Thus, due to the simplicity of using one method instead of an ensemble of methods, we decided that the tuned K-Nearest Neighbors model is the best for our use case.

We also learned the importance of data preprocessing. Feature engineering, encoding strategies, and model hyperparameter tuning significantly influenced the model's performance. User profiling, including features such as the number of months since the PIN was changed, account age, and various user activity metrics, was also very influential in the model's performance. The cyclical encoding of time-related features, such as month, day, and hour, using sine and cosine transformations has proven effective in capturing the temporal patterns in the data.

There are many ways to improve our model in the future. A few are outlined below.

1. Time Series:
   - ➔ Further exploration into advanced time series analysis techniques, such as autoregressive models or recurrent neural networks (RNNs), could be undertaken to capture more intricate temporal dependencies and finer temporal structures, especially in intra-day or intra-month patterns. Additionally, incorporating sliding window approaches or attention mechanisms might enhance the model's sensitivity to evolving fraud patterns over time.

2. User Profiling:
   - ➔ User profiling has proven essential, but there might be latent user behavior patterns not captured by explicit features. Unsupervised learning approaches could provide a more nuanced understanding of user behavior. Behavioral biometrics or recurrent neural networks could be investigated for their potential in creating more sophisticated user profiles. Employing clustering or anomaly detection algorithms to uncover hidden patterns within user behavior that may not be apparent in traditional feature engineering could also be explored.

3. Deep Learning:
   - ➔ The current work focused on traditional machine learning models, achieving strong performance. However, the inclusion of deep learning models might offer additional capabilities in capturing complex, nonlinear, or more intricate relationships within the data. The integration of deep learning architectures, such as deep neural networks or convolutional neural networks (CNNs), could be explored to leverage hierarchical feature representations and potentially uncover hidden patterns in the data. Transfer learning, using pre-trained models on related domains, could also be investigated to enhance model generalization or by incorporating the use of pre-trained models on related fraud detection datasets or other financial domains to leverage knowledge transfer.

4. Data and Model Investigation:
   - ➔ Continuous investigation into data preprocessing techniques and feature engineering methods, along with experimenting with different encoding strategies for categorical variables, can further refine the model's ability to extract meaningful information from the data. Ongoing model investigation, including robustness testing and interpretability studies, will contribute to building more reliable and trustworthy fraud detection systems.

5. Incremental Learning:
   - ➔ Develop mechanisms to detect and adapt to concept drift, ensuring the model remains effective as fraud patterns evolve over time.

- ➔ Investigate online learning approaches that update the model with new data points without retraining on the entire dataset.

6. Adversarial Training:

- ➔ Explore the use of Generative Adversarial Networks (GANs) to generate synthetic adversarial samples, helping the model become more robust against adversarial attacks.
- ➔ Investigate state-of-the-art techniques designed explicitly to enhance model robustness against adversarial manipulations.

In summary, the future work outlined above aims to advance the credit card fraud detection system by addressing nuanced aspects of time series analysis, user profiling, deep learning integration, and ongoing model investigation. Continuous exploration and adaptation will be paramount for staying at the forefront of fraud detection capabilities in the dynamic landscape of financial transactions.

7. References

Akin, J. (2020, June 23). *What are the Different Credit Score Ranges?* https://www.experian.com/blogs/ask-experian/infographic-what-are-the-different-scoring-ranges/

Alammar, A., Al Moayed, Y., & Ahmed Algeelani, N. (2022). Transaction fraud detector using KNN in deep learning. *International Journal of Novel Research in Computer Science and Software Engineering*, *9*(3), 16–23. https://doi.org/10.5281/zenodo.7365019

AlEmad, M. (2022). *Credit Card Fraud Detection Using Machine Learning* [Rochester Institute of Technology]. https://scholarworks.rit.edu/cgi/viewcontent.cgi?article=12455&context=theses

Alenzi, H., & Aljehane, N. (2020). Fraud Detection in Credit Cards using Logistic Regression. *International Journal of Advanced Computer Science and Applications*, *11*(12), 540–551.

Alghamdi, B., & Alharby, F. (2019). An Intelligent Model for Online Recruitment Fraud Detection. *Journal of Information Security*, *10*(3), Article 3. https://doi.org/10.4236/jis.2019.103009

Altman, E. R. (2019). *Synthesizing Credit Card Transactions*. arxiv.org. https://arxiv.org/abs/1910.03033

AnAj, A. A. (2007). *Example of k-nearest neighbour classification*. Own work. https://commons.wikimedia.org/wiki/File:KnnClassification.svg

Anusha, D. C., Lalitha, B. M., Sravya, B., Bindu, V. S., & Omkanth, T. (2023). Credit Card Fraud Detection Using Random Forest and Cart Algorithm. *International Journal of Scientific Research in Science and Technology*, *10*(2), 226–231. https://doi.org/10.32628/IJSRST52310236

Brownlee, J. (2020, June 4). How to Perform Feature Selection With Numerical Input Data. *MachineLearningMastery.Com*. https://machinelearningmastery.com/feature-selection-with-numerical-input-data/

Brownlee, J. (2020, May 24). Recursive Feature Elimination (RFE) for Feature Selection in Python. *MachineLearningMastery.Com*. https://machinelearningmastery.com/rfe-feature-selection-in-python/

Brownlee, J. (2021, January 27). *Undersampling Algorithms for Imbalanced Classification*. MachineLearningMastery.com. https://machinelearningmastery.com/undersampling-algorithms-for-imbalanced-classification/

Cherif, A., Badhib, A., Ammar, H., Alshehri, S., Kalkatawi, M., & Imine, A. (2023). Credit card fraud detection in the era of disruptive technologies: A systematic review. Journal of King Saud University - Computer and Information Sciences, 35(1), 145–174. https://doi.org/10.1016/j.jksuci.2022.11.008

Chung, J., & Lee, K. (2023). Credit Card Fraud Detection: An Improved Strategy for High Recall Using KNN, LDA, and Linear Regression. *Sensors*, *23*(18), Article 18. https://doi.org/10.3390/s23187788

Devi, D., Biswas, Saroj. K., & Purkayastha, B. (2019). A Cost-sensitive weighted Random Forest Technique for Credit Card Fraud Detection. *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 1–6. https://doi.org/10.1109/ICCCNT45670.2019.8944885

Egan, J. (n.d.). *Credit Card Fraud Statistics*. Bankrate. Retrieved September 22, 2023, from https://www.bankrate.com/finance/credit-cards/credit-card-fraud-statistics/

Feature-engine documentation—1.3.0. (n.d.). Retrieved October 29, 2023, from https://feature-engine.trainindata.com/en/1.3.x/index.html

Gajawada, S. K. (2023, August 16). *Chi-Square Test for Feature Selection in Machine learning*. Medium. https://towardsdatascience.com/chi-square-test-for-feature-selection-in-machine-learning-206b1f0b8223

GeeksforGeeks. (2023, September 13). *Naive Bayes Classifiers*. GeeksforGeeks. https://www.geeksforgeeks.org/naive-bayes-classifiers/

Guhanesvar. (2021, June 27). Feature Selection Based on Mutual Information Gain for Classification and Regression. *Medium*. https://guhanesvar.medium.com/feature-selection-based-on-mutual-information-gain-for-classification-and-regression-d0f86ea5262a

Imbalanced-learn documentation—Version 0.11.0. (n.d.). Retrieved October 29, 2023, from https://imbalanced-learn.org/stable/

Khatri, S., Arora, A., & Prakash Agrawal, A. (2020). *Supervised Machine Learning Algorithms for Credit Card Fraud Detection: A Comparison*. 680–683. https://doi.org/doi.org/10.1109/Confluence47617.2020.9057851

Kumar, Y., Saini, S., & Payal, R. (2020, October 18). Comparative Analysis for Fraud Detection Using Logistic Regression, Random Forest and Support Vector Machine. Available at SSRN: https://ssrn.com/abstract=3751339

Madhumitha, B., & Parthipan, V. (2023). A Novel Approach For Detecting Malicious Activities In Credit Card Transactions Using K-Nearest Neighbour Algorithm To Improve Accuracy And Compared With Gaussian Naive Bayes Algorithm. *European Chemical Bulletin*, *12*, 3559–3566.

Mahajan, T., Singh, G., Bruns, G., Bruns, G., Mahajan, T., & Singh, G. (2021, March). An experimental assessment of treatments for cyclical data. In Proceedings of the 2021 Computer Science Conference for CSU Undergraduates, Virtual (Vol. 6, p. 22).

Makki, S., Haque, R., Taher, Y., Assaghir, Z., Hacid, M.-S., & Zeineddine, H. (2018). *A Cost-Sensitive Cosine Similarity K-Nearest Neighbor for Credit Card Fraud Detection*. 42–47. https://ceur-ws.org/Vol-2343/paper10.pdf

Markus, Anneliese & Roche, Amos & Ngan, Chun-Kit & Cheung, Yin-Ting & Prifti, Kristi. (2022). A Prognostic Machine Learning Framework and Algorithm for Predicting Long-term Behavioural Outcomes in Cancer Survivors. 10.5220/0010893700003123.

Matplotlib.pyplot—Matplotlib 3.8.0 documentation. (n.d.). Retrieved October 29, 2023, from https://matplotlib.org/stable/api/pyplot_summary.html

Mehbodniya, A., Alam, I., Pande, S., Neware, R., Rane, K. P., Shabaz, M., & Madhavan, M. V. (2021). Financial Fraud Detection in Healthcare Using Machine Learning and Deep Learning Techniques. *Security and Communication Networks*, *2021*, e9293877. https://doi.org/10.1155/2021/9293877

Merchant Category Codes (MCC)—ISO 18245:2003—Classification.codes. (n.d.). Retrieved October 29, 2023, from https://classification.codes/classifications/industry/mcc/

Michaelg2015. (2015). *English: Graph of a logistic regression curve showing probability of passing an exam versus hours studying*. Own work. https://commons.wikimedia.org/wiki/File:Exam_pass_logistic_curve.jpeg

Mishra, K. N., & Pandey, S. C. (2021). Fraud Prediction in Smart Societies Using Logistic Regression and k-fold Machine Learning Techniques. *Wireless Personal Communications*, *119*(2), 1341–1367. https://doi.org/10.1007/s11277-021-08283-9

Mqadi, N.M., Naicker, N., & Adeliyi, T. (2021). Solving Misclassification of the Credit Card Imbalance Problem Using Near Miss. Mathematical Problems in Engineering, 2021, 16 pages. https://doi.org/10.1155/2021/7194728.

Nami, S., & Shajari, M. (2018). Cost-sensitive payment card fraud detection based on dynamic random forest and k-nearest neighbors. *Expert Systems with Applications*, *110*, 381–392. https://doi.org/10.1016/j.eswa.2018.06.011

Ogundokun, R. O., Misra, S., Fatigun, O. J., & Adeniyi, J. K. (2022). Naïve Bayes Based Classifier for Credit Card Fraud Discovery. In M. Themistocleous & M. Papadaki (Eds.), *Information Systems* (pp. 515–526). Springer International Publishing. https://doi.org/10.1007/978-3-030-95947-0_37

Pandas—Python Data Analysis Library. (n.d.). Retrieved October 29, 2023, from https://pandas.pydata.org/

Pei, K. (2017, April 13). *How is Naive Bayes a Linear Classifier?* [Forum post]. Cross Validated. https://stats.stackexchange.com/q/142215

Poongodi, K., & Kumar, D. (2021). Support vector machine with information gain based classification for credit card fraud detection system. Int. Arab J. Inf. Technol., 18(2), 199-207.

Ranganathan, S., Gribskov, M., Nakai, K., Schönbach, C., & Berrar, D. (2019). Bayes' Theorem and Naive Bayes Classifier. In *Encyclopedia of Bioinformatics and Computational Biology* (Vol. 1, pp. 403–412). essay, Elsevier Inc.

Russell, S., Norvig, P., Chang, M.-W., Devlin, J., Dragan, A., Forsyth, D., Goodfellow, I., Malik, J. M., Mansinghka, V., Pearl, J., & Wooldridge, M. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.

Scikit-learn: Machine learning in Python—Scikit-learn 1.3.2 documentation. (n.d.). Retrieved October 29, 2023, from https://scikit-learn.org/stable/index.html

Sriram Sasank, J. V. V., Ram Sahith, G., Abhinav, K., & Belwal, M. (2019). *Credit Card Fraud Detection Using Various Classification and Sampling Techniques: A Comparative Study*. 1713–1718. https://doi.org/doi.org/10.1109/ICCES45898.2019.9002289

Wang, T., & Zhao, Y. (2022). *Credit Card Fraud Detection using Logistic Regression*. 301–305. https://doi.org/doi.org/10.1109/BDICN55575.2022.00064

Zhang, D., Bhandari, B., & Black, D. (2020). Credit Card Fraud Detection Using Weighted Support Vector Machine. Applied Mathematics, 11, 1275-1291. doi:10.4236/am.2020.1112087