Adina Palayoor
Jeffrey Chan
Taya Yakovenko
Ethan Rudometkin
Nathan Willemsen
4/27/23

MA 4635 Final Report

This dataset looks at financial transactions made using a credit or debit card. Each row in the dataset represents a single transaction and includes information such as the distance of the transaction from the user's home and the user's last transaction, the ratio of the transaction amount to the median purchase price which gives an idea of how the transaction amount compares to the user's typical spending behavior. The dataset also includes variables indicating whether the transaction is a repeat from a previous retailer, whether a chip or PIN number was used in the transaction, whether the transaction was made online, and whether the transaction has been identified as potentially fraudulent.

There are several questions that could be asked about the given dataset

1. Can we determine the input features that are affecting our overall accuracy the most?
2. How do we limit the number of times that we are not predicting fraud when we should be?
3. How do we determine the best classification model to predict credit card fraud?

There were several methods we used to ensure that our model was using correct and balanced data. We started by reading in the data as a csv into our python file. We then stored the data as two X and Y dataframes where X stored the input features and Y stored the fraud indication. From here we wanted to make sure we created three sets: a training set, a validation set and a test set. To do this, we used the train_test_split package from sklearn to separate out the dataset as 80% training data and 20% validation data. After this split we did another split of the

training set into a training set of 80% of the original training data and 20% test data. We did this so that we could test our model on the test data first and then run the data again through the validation set to ensure there is no overfitting. The final step in preprocessing the data was using the SMOTE package from imblearn to balance our data. We ran our input and output training data through SMOTE to get the resampled dataset that we used within our Decision Tree and KNN models. For Gradient Descent, we had a slightly different process for our data that we will discuss later.

Our first approach to identify fraudulent transactions was to use the K-Nearest Neighbors (KNN) algorithm. K-Nearest Neighbors (KNN) is a non-parametric algorithm that is commonly used for classification problems. It works by finding the K nearest data points in the training set to a new data point and classifying it based on the most common class among its neighbors. One of the benefits of using KNN is that it can handle complex decision boundaries and is relatively easy to implement. In the context of credit card fraud detection, KNN is a good choice because it can identify patterns in the data that might not be immediately apparent and can handle high-dimensional data with many features. KNN is an efficient algorithm, which makes it a practical choice for large datasets like the one we are working with. It provides a good starting point for identifying fraudulent transactions in the credit card dataset.

Our team used the KNeighborsClassifier() function to create a KNN classifier, which we fit to the training data using the fit() function. We then predicted the labels of the test data using the predict() function, and evaluated the performance of the model using various classification metrics such as accuracy, precision, recall, F1 score, and confusion matrix. The evaluation results of the KNN model show that it is able to predict fraudulent transactions with high accuracy. The accuracy score of 0.974 indicates that the model can correctly identify 97.4% of the transactions.

The precision score of 0.778 indicates that out of all the transactions predicted as fraudulent, only 77.8% were actually fraudulent. Recall score of 0.987 shows that out of all the actual fraudulent transactions in the test data, the model was able to correctly identify 98.7% of them. The F1 score of 0.870, which is a weighted average of precision and recall, suggests that the overall performance of the model is good. When looking at the confusion matrix, we can see that there were 3,957 false positives, indicating that some legitimate transactions were flagged as fraudulent. The model performed well in identifying fraudulent transactions, with only 182 false negatives, meaning that most fraudulent transactions were correctly identified.

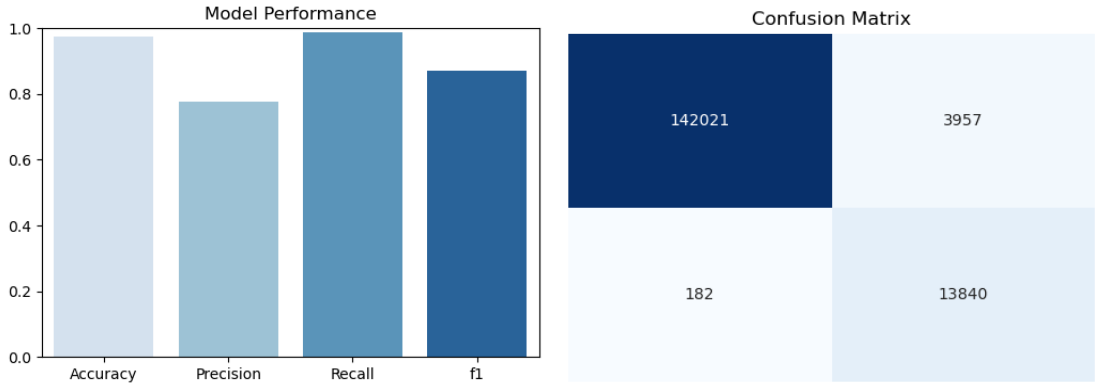


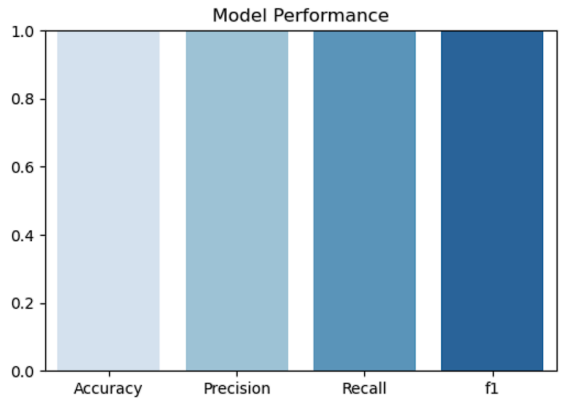
Figure 1. Bar plot and heatmap of evaluation metrics of KNN model

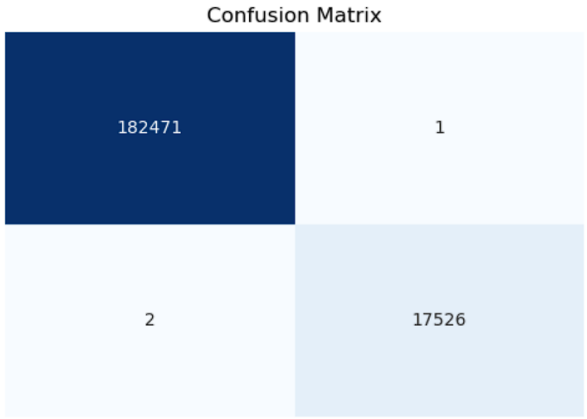
In addition to the models mentioned above, we also implemented a gradient descent classification model. This model allows us to iteratively optimize the weights for each of the parameters as we train the model by calculating the gradient at each step (determined by the learning rate) and decreasing it to reach the minimum of the loss function. Despite the fact that this algorithm may be computationally expensive, it usually does a great job in finding the most optimal solution without overfitting the data set too much.

The data set was separated into testing(25%) and training(75%) sets which were then normalized to increase accuracy of the model. Sincere there were only 8 prediction variables, they were all used in training and testing the model. The model was created using sklearn and pandas packages to sort the data and train the data respectively. The f1 score for this model was 0.7539 which is relatively low. Based on the data set used, the f1 score can be explained by the fact that the data is unbalanced. We had a lot more incidents where the transactions were not fraudulent which made it hard to predict fraud. In order to model's performance, we can try to balance our dataset using the bootstrap method. Finally, we can increase the performance of this model by running stochastic gradient descent using the partial_fit method in SGDClassifier under the sklearn package. Stochastic gradient descent runs the gradient parameter optimization on a subset of the data set thus being able to extract a better prediction result.

The last model we tried was a decision tree classifier implemented in python. We chose a decision tree classifier because the dataset had some outliers in the distance from home, distance from last transaction, and ratio to median purchase price columns and a decision tree classifier can easily separate outliers into their own tree if they heavily indicate fraud. For example, most transactions are close to home and a normal price, so if a transaction We first set aside 20% of the data as a validation dataset and then split the remaining data into test (20%) and train (80%) data with the target variable being whether or not the transaction was fraudulent. Then we used the DecisionTreeClassifier() method from the sklearn library to train a decision tree classifier using the training dataset and predicted fraud in the test dataset to test the model. Then we checked the validation dataset to find the Accuracy, precision, recall, and F1 scores of the model to measure its effectiveness. The decision tree was very good at predicting whether or not there

was fraud and had a 99.9% effectiveness in all of the evaluation metrics which is an extremely good model to predict the fraud.





In the table below, we outline the three models we tried and their corresponding accuracy, precision, recall and F1 score. Our focus in the project was being able to limit the number of false negatives meaning that we limit how many times we are not predicting fraud when we should be. This means that we really focused on recall when trying to understand the model's overall effectiveness.

metrics	Decision Tree	KNN	Gradient Descent

Accuracy	0.9999	0.9741	0.9638
Precision	0.9998	0.7777	0.8705
Recall	0.9996	0.987	0.6855
F1 score	0.9997	0.8699	0.767

Based on our table, Both the Decision Tree and KNN were the best for the recall score as they were both close to 100%. To understand which of these was better we then moved to our precision score as well as the F1 score. In both of these cases, the Decison tree was significantly more accurate at predicting fraud.

For this reason, we decided to conclude that the best model to use for predicting fraudulent credit card activity is a decision tree. While there is a question on if the model is overfitting due to its very high accuracy metrics, we still believe that given completely new credit card transaction data, the decision tree would be able to predict fraud with high accuracy.

Overall, the credit card transaction dataset provided an good opportunity to explore different classification models and techniques for identifying fraudulent transactions. The Decision Tree model proved to be the most effective choice for this dataset, with a high accuracy score of 0.9999 along with other good performance metrics. The KNN classification model also showed potential but had a relatively low precision score of 0.7777, meaning this model wasn't what we were looking for in limiting the times that we are not predicting fraud when we should be. Further improvements to the models could be made by balancing the dataset and running stochastic gradient descent. Preprocessing techniques such as using SMOTE for balancing the data and splitting the data into training, validation, and test sets were also essential to ensure

model accuracy and avoid overfitting. Ultimately, this project demonstrates the value of machine learning in identifying fraudulent transactions and highlights the importance of choosing the right model to identify fraudulent transactions.