CS 471 Team 2 Final Project Final Report

Names:
Shane Dyrdahl, Alastair Raymond, Riley Saliba, Dax Taraleskof

Date:
06 December 2024

Code link:
https://github.com/daxmictar/CSUSM-CS471-FinalProject

Motivation:
Bank fraud is a growing problem. To address this, we seek to create a model which can determine if a bank transaction is fraudulent based on data collected at the time. Our model will have a fast inference time such that it can be run on new transactions as they are processed. It is important for our model to produce as few false negatives as possible. As such, recall will be the primary metric during evaluation and testing. While accuracy and f-score are important as well, recall will be prioritized.

Prior Research:
Prior research on this problem has been documented on Kaggle (https://www.kaggle.com/code/kishankumar2110/fraudulent-transactions-predict-with-99-accuracy). Additional work referenced includes an academic paper as well (https://philpapers.org/rec/MEGFFT-2).

The first paper utilized a logistic regression model for classification. This contrasts to the second paper which experimented with several classification models including decision tree and random forest.

Their results had multiple different machine learning models and even some that we didn't even consider using such as MLP Regressor or Gaussian NB. They had no rules-based approach but to compare with the machine learning models that we did use, their results are presented below.

Model Name	*Accuracy*	*Precision*	*Recall*	*F1*
Decision Tree Classifier	99.958%	99.945%	99.971%	99.958%
Random Forest Classifier	99.950%	99.950%	99.950%	99.950%

It was not explicitly stated in the article if the results shown were done on the training or testing data as it says that the data was split by 60-20-20 for training, validation, and testing. Though it can be assumed that the table with the balanced dataset was done with training data, while the

unbalanced dataset table displays the testing dataset as one only balances the training set, not testing.

For our decision tree model, we achieved a recall of 99.65%, while the random forest one had a recall of 97.65%. So overall,

Goals:
As previously stated, our main goal is to classify transactions as fraudulent or not fraudulent. Our models will differ from prior research by prioritizing recall over other scoring metrics. In addition, we wish to produce a final model which will yield a low inference time. This is so it may be used in a production environment to complete classification.

Dataset:
Our dataset is the fraudulent transactions dataset (https://www.kaggle.com/datasets/vardhansiramdasu/fraudulent-transactions-prediction). The dataset is composed of a series of simulated transactions which mimic actual bank transactions.

Here are the features included in our dataset:

- step: The time step of the transaction in respect to the simulation
- type: The type of transaction (cash-in, cash-out, debit, payment, transfer).
- amount: The total amount of the transaction.
- nameOrig: The ID of the customer who initiated the transaction.
- oldbalanceOrg: The account balance before the transaction for the originator.
- newbalanceOrig: The updated account balance after the transaction for the originator.
- nameDest: The ID of the customer receiving the transaction.
- oldbalanceDest: The recipient's account balance before the transaction.
- newbalanceDest: The updated account balance after the transaction for the recipient.
- isFraud: Indicates whether the transaction is actually fraudulent
- isFlaggedFraud: Flags transactions that are considered illegal; specifically, this applies to attempts to transfer more than 200,000 in a single transaction.

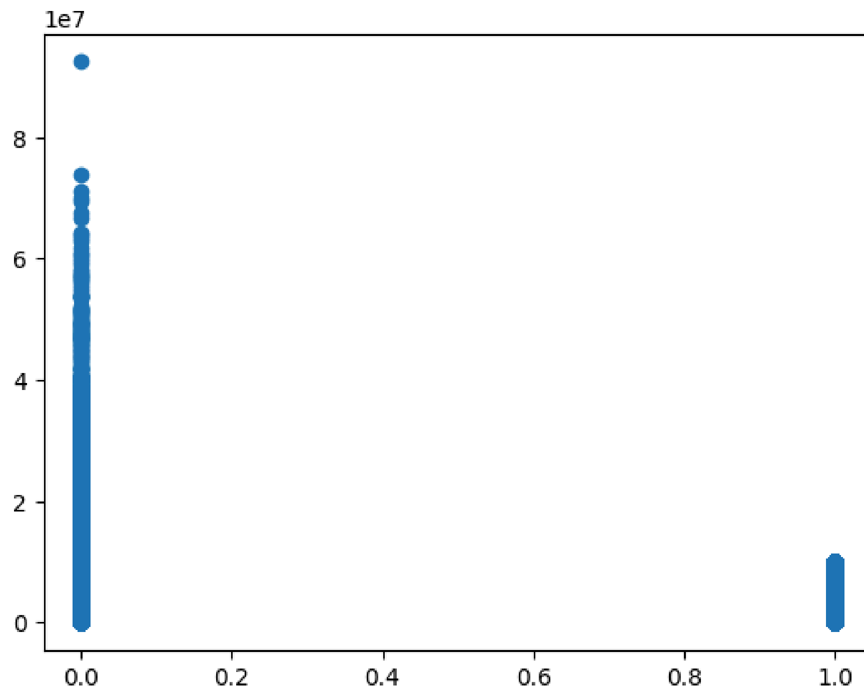
We excluded step and isFlaggedFraud from our model. This is as they are specific to the simulation which generated the dataset and are not relevant for classification.

There were no rows which were missing features in our dataset.

The dataset is highly imbalanced, consisting of approximately 6,084,104 non-fraudulent transactions compared to just 6,485 fraudulent transactions. As a result, we needed to apply oversampling to ensure that fraudulent transactions are adequately represented.

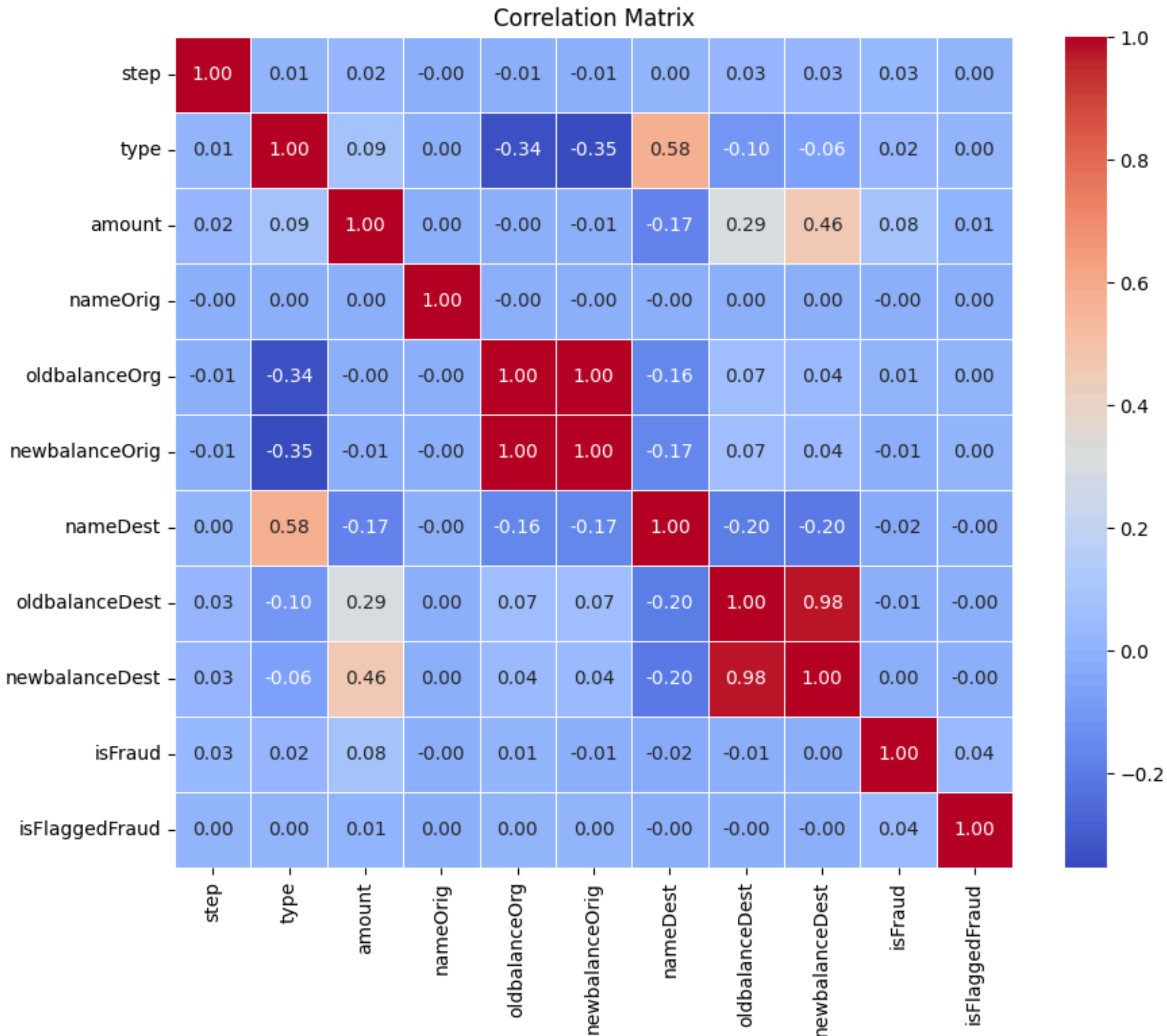
Exploratory Data Analysis:

We first examined the dataset through a scatter plot. In the chart shown below, the x-axis represents if a transaction is fraudulent (0 or 1). The y-axis represents the amount of the transaction. The vast difference in amount between a fraudulent and non-fraudulent transaction can easily be seen.



As mentioned earlier, the dataset was initially highly imbalanced, with the graph illustrating that the majority of the data consists of 0s (non-fraudulent transactions) while a small portion comprises 1s (fraudulent transactions).

We then examined the dataset through a correlation matrix.



A strong correlation between the features newbalanceOrig and oldbalanceOrg can be seen. This is also true of nameDest and type.

Pre-processing:

Since we are using a machine learning model, numerical inputs are preferred. Therefore, we apply encoding to convert the categorical variables into unique integers, as these are the only features represented as strings. We use the label encoder object provided by sklearn.

```
# encode strings
label_encoder = preprocessing.LabelEncoder()
one_hot_encoder = preprocessing.OneHotEncoder()
dataset['type'] = label_encoder.fit_transform(dataset['type'])
dataset['nameOrig'] = label_encoder.fit_transform(dataset['nameOrig'])
dataset['nameDest'] = label_encoder.fit_transform(dataset['nameDest'])
```

We separate the features from the target variable so that the independent variables (X) can be used for making predictions, while the dependent variable (Y) represents the outcome which the model aims to predict. In this instance, the variables in X are used to predict whether a transaction is fraudulent or not.

```
X = dataset[['type', 'amount', 'nameOrig', 'oldbalanceOrg',
'newbalanceOrig', 'nameDest', 'oldbalanceDest', 'newbalanceDest']]
y = dataset[['isFraud']]
```

We then split the dataset into training and testing.

```
# split dataset into training and testing
training_X, testing_X, training_y, testing_y = train_test_split(X, y,
test_size = 0.20)
```

The data is split again into training and evaluation. The result is three sets of data (60% training, 20% evaluation and 20% testing).

```
training_X, evaluation_X, training_y, evaluation_y =
train_test_split(training_X, training_y, test_size = 0.20)
```

As the dataset is very imbalanced, we oversample the training data using SMOTE from the imbalanced learn library.

```
temp_X, temp_y = SMOTE(random_state=130).fit_resample(training_X,
training_y)
training_X = temp_X
training_y = temp_y
```

Algorithm 1 (Rules-based approach):

We originally intended to use the statistical mean and/ or median of the transaction set for classification. However, we settled on using a fixed threshold, as this offered better performance. The median value did provide insights on the dataset which allowed for initial experimentation in regards to the threshold value.

The classification from the rules based approach can be seen below.

```
def evaluate(threshold, data, actual):
    false_negative_count = 0
    true_positive_count = 0
    merged_data = data.copy()
```

```
    merged_data.insert(8, "isFraud", actual, True)
    potential_fraud = merged_data.loc[(merged_data['amount'] < threshold)]
    actual_fraud = potential_fraud.loc[(potential_fraud['isFraud'] == 1)]
    not_fraud = potential_fraud.loc[(potential_fraud['isFraud'] == 0)]
    true_positive_count = len(actual_fraud)
    total_fraud_count = len(merged_data.loc[(merged_data['isFraud'] ==
1)])
    false_negative_count = total_fraud_count - len(actual_fraud)
    false_positive_count = len(not_fraud)

    # calculate recall metric
    recall = (true_positive_count / (true_positive_count +
false_negative_count))
    print("Recall: " + str(recall))

    # calculate precision metric
    precision = (true_positive_count / (true_positive_count +
false_positive_count))
    print("Precision: " + str(precision))

    # calculate F1-score
    print("F1-Score: " + str(2 * ((precision * recall) / (precision +
recall))))
```

The algorithm first reads the supplied data and splits it into three different tables. The first table (potential_fraud) includes transactions which match the rule (amount < threshold). The second table (actual_fraud) includes transactions which are marked as fraud (isFraud = 1) out of potential_fraud. Finally, the third table includes transactions which are not fraud (isFraud = 0), but are included in the potential_fraud table.

The amount of true positives, false negatives and false positives are derived from the lengths and operations thereof on these tables.

Ultimately, the algorithm simply classifies any transactions which are below the supplied threshold as fraudulent. Work on determining which threshold value to use is documented in "Experiments".

Positive attributes of rules-based approach:

- Simple and Interpretable: The approach is straightforward to implement, involving a simple comparison between the transaction amount and an adjustable threshold. It is also easy to explain why a transaction is flagged as fraudulent.
- Computationally inexpensive: Although our dataset is large, the process only requires a basic comparison, making it efficient and speedy during execution.
- Good recall score: The algorithm performs well in terms of recall (> 0.96).

Negatives attributes of rules based approach:

- Potential to Miss Fraudulent Transactions: Detecting fraud can be more nuanced than simply comparing the transaction amount to a set threshold. There may be instances where a transaction amount is below the threshold, but is still fraudulent. This would result in a false negative.
- Lower Precision: Initial evaluation yielded a precision score of 0.4921. This means less than 50% of transactions which were classified as fraudulent were actually fraudulent. The remaining amount were legitimate transactions which were determined to be fraudulent. As such, the rules-based approach results in a large amount of false positives.

Algorithms 2 (machine learning approach):

For our second algorithm, we opted for a decision tree as it made sense for determining if a transaction was fraud versus not fraud, because of a decision tree's binary classification. Additionally, a decision tree model was one of the algorithms used in the prior research we found.

The algorithm first stores the best hyperparameters that had the best performance from cross validation and also stores the best recall score.

From there, it stores the decision tree model with the best hyperparameters in best_model and evaluates the best model by making predictions on the X_dataset.

We used a hyperparameter grid for decision tree model tuning and use max_depth and min_samples_split as our hyperparameters with only a single combination of max_depth=4 and min_samples_split=2, as our dataset is very large and this was the minimum we could train on without maxing out the cpu.

grid_search_dt stores the best performing hyperparameters based on the recall and this will be used further when actually evaluating the model on the training data.

We then evaluate the model on the evaluation set so we can see if we need to tune the model and finally we evaluate the model now with the testing set for a final unbiased evaluation.

When utilizing dt_evaluate on the evaluation and testing dataset, it prints us the recall of the best model, a classification report (precision, recall, F1_score, and support) on isNotFraud and isFraud classes, and a confusion matrix.

Positive attributes of DT:

- High interpretability: The visual flow chart-like structure of a decision tree makes it easy to understand why a transaction was classified as fraud or not as the path it took to reach its classification can be understood

Negatives attributes of DT:

- Overfitting: If the tree was too deep, it may have a difficult time with unseen data as the model has already been tuned with so many specific patterns

Algorithm 3 (machine learning approach):

For our third algorithm, we chose to use a random-forest model. We initially thought we would see an improvement in performance over the previous two approaches. This is especially true in regards to decision tree models, as a random-forest model uses many decision trees.

We built our model while reserving n_estimators for tuning.

Further discussion on tuning of these values is documented in “Experiments”.

A RandomForestClassifier was created, as our model will be used for classification.

GridSearchCV was used to evaluate various values for the aforementioned hyperparameters during tuning. The scoring parameter was set to “recall”, as our primary scoring method is recall.

Finally, the model was fitted onto the training data and scoring metrics were produced using the rf_evaluate method.

Positive attributes of Random Forest Approach:

- Improved recall scoring due to larger size of model from multiple decision trees

Negative attributes of Random Forest Approach:

- Significantly increased training time
- Increased inference time
- Larger model size

Experiments:

Rules Based Tuning

We tried multiple different threshold values before ending up with 100,000 (mode).

Here we used median:

```
Median = 172383.67472022248
Maximum = 71172480.42
Mode = 10000000.0
C:\Users\pinoy\AppData\Local\Temp\ipykernel_13288\277321585.py
  mode = float(training_X[['amount']].mode().iloc[0])
```

```
evaluate(172383.67472022248, evaluation_X, evaluation_y)
```

✓ 0.3s

```
Recall: 0.30721304443639913
Precision: 0.30718493896554744
F1-Score: 0.30719899105813464
```

Then we tried max:

```
Median = 172383.67472022248
Maximum = 71172480.42
Mode = 10000000.0
C:\Users\pinoy\AppData\Local\Temp\ipykernel_13288\27732158
  mode = float(training_X[['amount']].mode().iloc[0])
```

```
evaluate(71172480.42, evaluation_X, evaluation_y)
```

✓ 0.4s

```
Recall: 1.0
Precision: 0.49988000487849016
F1-Score: 0.6665599958031135
```

Mode:

```
Median = 172383.67472022248
Maximum = 71172480.42
Mode = 10000000.0
C:\Users\pinoy\AppData\Local\Temp\ipykernel_13288\
  mode = float(training_X[['amount']].mode().iloc[
```

```
evaluate(10000000, evaluation_X, evaluation_y)
```

✓ 0.4s

```
Recall: 0.966543822506257
Precision: 0.49158243935633655
F1-Score: 0.6517075817638599
```

double the mode:

```
Median = 172383.67472022248
Maximum = 71172480.42
Mode = 10000000.0
C:\Users\pinoy\AppData\Local\Temp\ipykernel_13288\2
  mode = float(training_X[['amount']].mode().iloc[0
```

```
evaluate(20000000, evaluation_X, evaluation_y)
```

```
✓ 0.4s
```

```
Recall: 1.0
Precision: 0.49991614488786285
F1-Score: 0.6665921246220571
```

From the results, we can see that using the max yields the strongest results, but having the highest transaction amount as the threshold would not exactly seem fair as we are seeing if the amount is lower than the threshold. Any amount below the highest would be detected as an anomaly, which works, but doesn't work in this context.

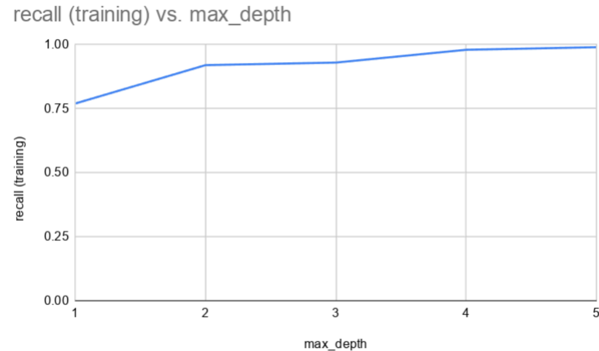
Instead, it made sense for us to use the mode, because we use the most common transaction amount as our threshold to detect any anomalies (amounts lower than the mode). This makes it the most fair in the context of fraud detection as we look at all the amounts to make it more fair for all recipients.

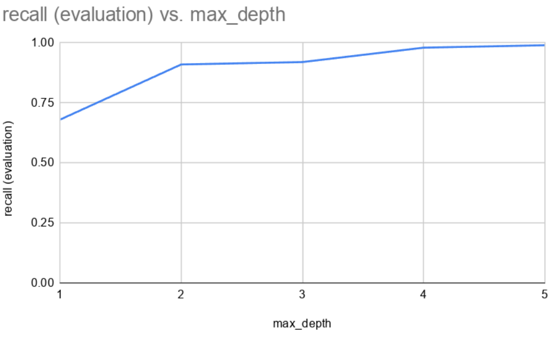
Using median as a threshold value would have been even more fair to customers but as clearly seen before, this resulted in the worst recall score out of all the values.

Decision Tree Tuning

We first approached tuning our decision tree model by modifying the value of max_depth. This hyperparameter sets the maximum size of the tree.

The various results of tuning max_depth can be seen below:



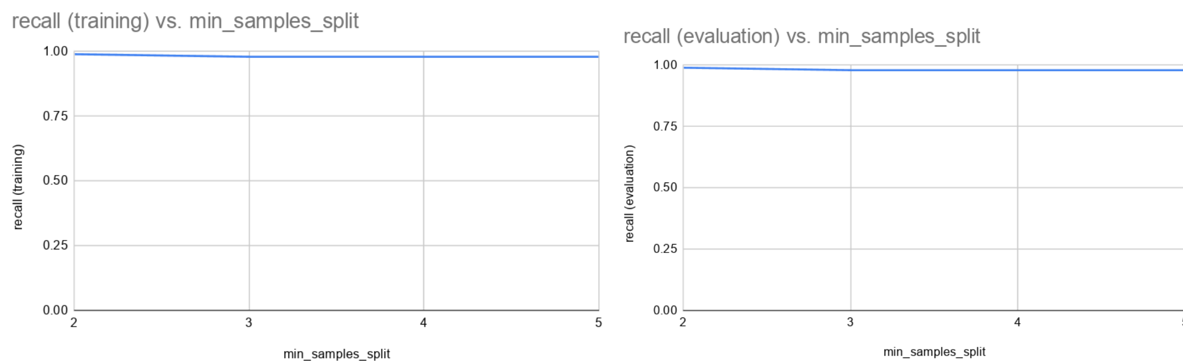


As one can see, lower values of max_depth produced reduced recall performance. This was the case on both the training and evaluation data. In addition, overfitting is present when max_depth is set to one.

From this experiment, we settled on setting max_depth to four, as it produced the best compromise between model size and performance.

Following max_depth, we progressed to tuning min_samples_split. This hyperparameter sets the minimum number of samples needed for an internal node to be split. Setting this value higher creates a tree which has less fine branching. This can aid in situations with overfitting.

The results of tuning this hyperparameter can be viewed below:



As seen, changing min_samples_split only had a negligible impact on model performance. From this, we set min_samples_split to two, as it produced the best recall on both training and evaluation data (0.99).

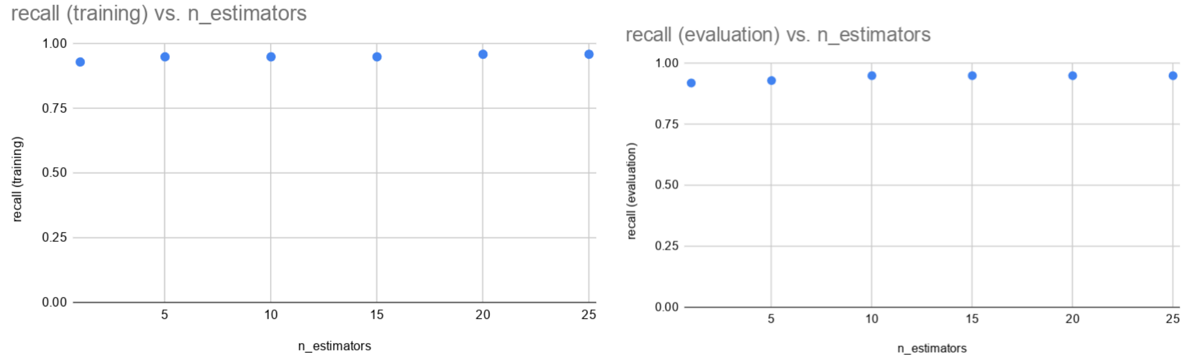
Random Forest Tuning

Tuning the random forest model was an exercise of patience.

Brief experimentation was first done on max_depth, however very similar results to that produced in tuning of the decision tree were yielded. As such, the value of max_depth was also set to four.

The sole hyperparameter we sought to investigate for tuning the random forest model was n_estimators. n_estimators sets the total number of decision trees used by the model. As one can imagine, when this hyperparameter is set to a high value, training time and inference time both increase.

The results of tuning n_estimators can be seen below.



As shown, vastly different values assigned to n_estimators did not produce dramatic improvement to recall. This was somewhat surprising to the team, as the advantage of random forest is in its plurality.

From these results, we decided to progress with a decision tree, as it provided better inference time, model size and recall.

Model Comparison and Results Analysis:

Note: Results from each model are shown below. Discussion follows after results.

Rules-based Approach

Training

```
>>> evaluate(10000000, training_X, training_y)
Recall: 0.9663871125271187
Precision: 0.4916622328930251З
F1-Score: 0.6517420649398029
```

Testing

```
>>> evaluate(10000000, testing_X, testing_y)
Recall: 0.9665689149560117
Precision: 0.0012962476462250072
F1-Score: 0.0025890232014497273
```

Decision Tree

Training

```
>>> dt_evaluate(training_X, training_y, grid_search_dt)
Best params: {'max_depth': 4, 'min_samples_split': 2}
Best recall from GridSearchCV tuning: 99.65%
recall of best estimator from GridSearchCV on test set for DT: 99.65%
              precision    recall  f1-score   support

  isNotFraud       1.00      0.96      0.98   4066833
     isFraud       0.96      1.00      0.98   4066833

    accuracy                           0.98   8133666
   macro avg       0.98      0.98      0.98   8133666
weighted avg       0.98      0.98      0.98   8133666

Confusion matrix (DT):
 [[3894105  172728]
 [  14060 4052773]]
```

Testing

```
>>> dt_evaluate(testing_X, testing_y, grid_search_dt)
Best params: {'max_depth': 4, 'min_samples_split': 2}
Best recall from GridSearchCV tuning: 99.65%
recall of best estimator from GridSearchCV on test set for DT: 99.53%
              precision    recall  f1-score   support

  isNotFraud       1.00      0.96      0.98   1270819
     isFraud       0.03      1.00      0.06      1705

    accuracy                           0.96   1272524
   macro avg       0.52      0.98      0.52   1272524
weighted avg       1.00      0.96      0.98   1272524

Confusion matrix (DT):
 [[1216665   54154]
 [      8    1697]]
```

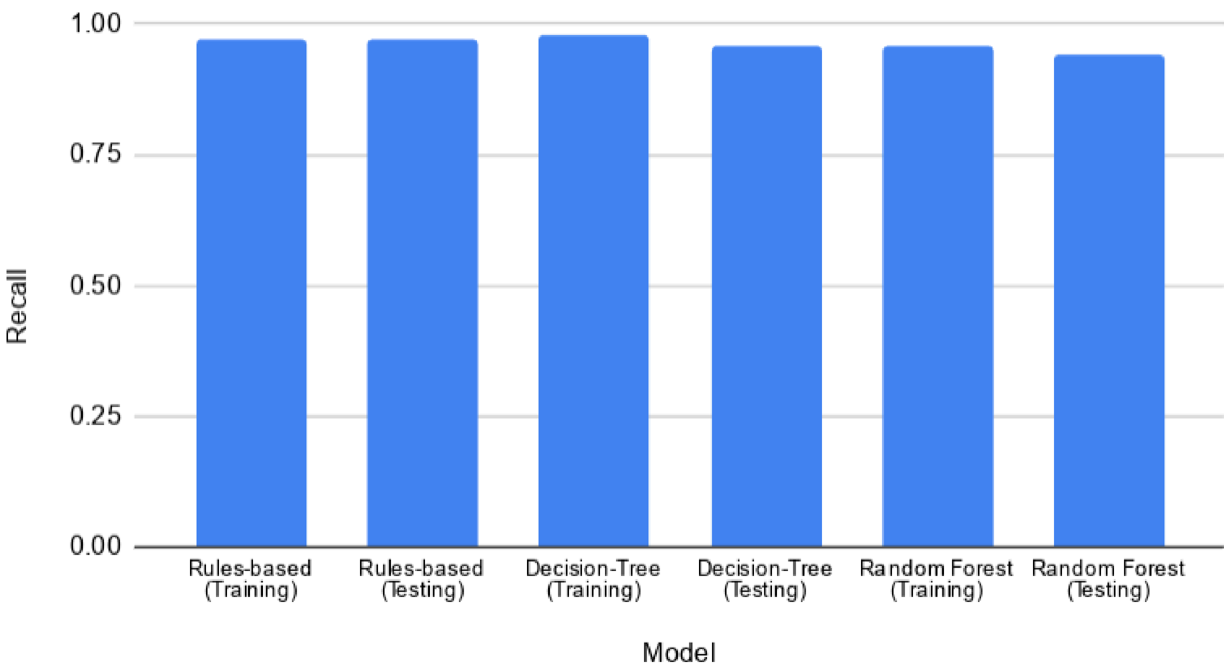
Random Forest

Training

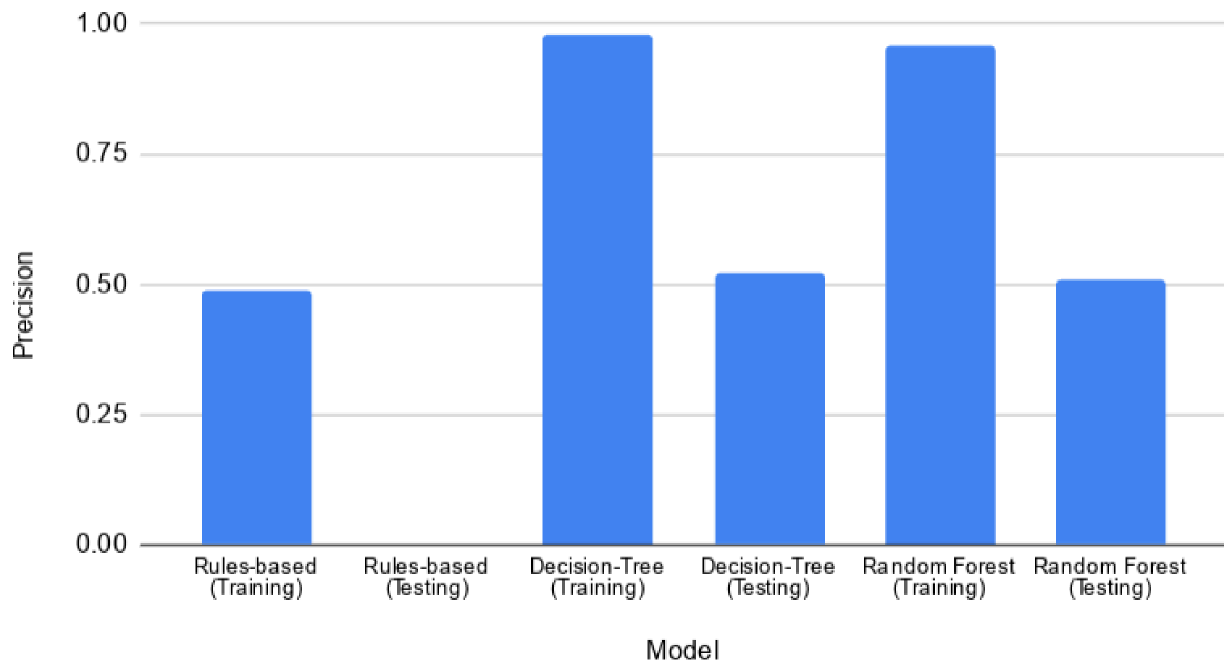
```
>>> rf_evaluate(training_X, training_y, grid_search_rf)
Best params: {'max_depth': 4, 'n_estimators': 25}
Best recall from GridSearchCV: 97.65%
recall of best estimator from GridSearchCV on test set for RF: 97.66%
              precision    recall  f1-score   support

  isNotFraud       0.98      0.94      0.96   4066833
     isFraud       0.94      0.98      0.96   4066833

    accuracy                           0.96   8133666
   macro avg       0.96      0.96      0.96   8133666
weighted avg       0.96      0.96      0.96   8133666

Confusion matrix (RF):
 [[3810803  256030]
 [  95324 3971509]]
```

Testing

```
>>> rf_evaluate(testing_X, testing_y, grid_search_rf)
Best params: {'max_depth': 4, 'n_estimators': 25}
Best recall from GridSearchCV: 97.65%
recall of best estimator from GridSearchCV on test set for RF: 95.48%
              precision    recall  f1-score   support

  isNotFraud       1.00      0.94      0.97   1270819
     isFraud       0.02      0.95      0.04      1705

    accuracy                           0.94   1272524
   macro avg       0.51      0.95      0.50   1272524
weighted avg       1.00      0.94      0.97   1272524

Confusion matrix (RF):
 [[1190993   79826]
 [     77    1628]]
```



As shown, our three approaches performed well on both the training and testing set. Although, this is only true when recall is considered.



All models display significant overfitting when measured by precision. In addition, performance on the training set is mixed as well. These statistics should be discounted, however, as our primary goal was to ensure that false negatives were not missed, thus prioritizing recall.

As previously discussed, random forest did not result in a significant performance improvement over a single decision tree. Due to this, the decision tree model would likely be the one we would move forward with.

Future improvements of this work would include additional tuning such that precision scoring would increase on the decision tree model. One can imagine in a production scenario, many transactions being flagged as fraud could impact the efficiency of bank operations.One can imagine in a production scenario, many transactions being flagged as fraud could impact the efficiency of bank operations.

Team Member Contributions:

Team Member	Contributions
Shane Dyrdahl	Decision tree experimentation, Random forest experimentation, runtime efficiency, dataset research, project ideas, Final report
Alastair Raymond	Rules-based approach design, Project proposal writeup, Rules-based approach implementation, data pre-processing, Final report, Random forest experimentation
Riley Saliba	Exploratory data analysis, Rules-based approach design, Rules-based approach experimentation, Progress report writeup, Final report
Dax Taraleskof	Decision tree design, Decision tree implementation, Random forest design, Random forest experimentation, Random forest implementation, Final report