Andrew Vu

Prof. McCreless

CIS 450

7 April 2023

Bank Application Fraud Detection

Introduction:

Cybercriminals have been using stolen or fake personal information to obtain loans or open bank accounts. With access to lines of credit, these bad actors can essentially “cash out” by maximizing the credit line without paying back. For example, one strategy by fraudsters is called “bust out” where they build up the credit line for multiple accounts before cashing out all at once (“Application Fraud”). This negatively impacts both consumers whose identities are stolen and financial institutions who suffer a loss in revenue. Specifically, the latter group was estimated to lose over $3.5 billion due to this scheme in 2021 (“Preventing Identity Fraud”).

The research question behind this analysis is to determine which factors of a bank account application most strongly indicate that the account was set up to commit fraud. By identifying the top factors, a financial institution could construct a more advanced model to proactively identify fraudulent accounts, adjusting the weights of variables based off their relative importance in this analysis. Depending on which variables are deemed to be important, the findings could also be incorporated into a model that pre-screens applicants prior to being granted a credit line. Both extensions of this analysis would thus help drive down fraud loss and reduce the number of stakeholders affected.

Problem Outline:

Using the fraudulent account variable as the target variable, the analysis will be conducted as a classification problem. Various techniques will be used to address the problem, such as multiple regression, decision trees, and random forest. Some models, such as regression and decision trees, are more prone to overfitting on training data so having a variety of models will allow for comparison of not only accuracy but other metrics such as AUC and F1-score. The most important variables from the best performing models would subsequently be identified.

Tableau will be used to create visualizations to assist in exploratory analysis while Python will be utilized to prepare data and construct the models. Regarding the former, a visual like the one below may be created to identify if there are any noticeable trends between different values or value buckets of a predictor and the target variable. This may assist in the feature selection process at the time of model creation.

Data Description:

The Bank Account Fraud Dataset Suite (NeurIPS 2022) was obtained from Kaggle as CSV files. 2.5 million records were initially sampled from a model that generated synthetic records before the final 1 million records were sampled. There are six variants in the suite, each with a different type of bias affecting the final sampling of records to allow for testing of the strength of various machine learning models. This analysis will focus on the base variant which does not introduce any form of bias. It should be noted that the dataset is extremely unbalanced, with only about 1.1% of records having been labeled as fraudulent. Fraud cases tend to be very infrequent in the real-world, so this metric directly reflects that.

Each row of the dataset represents an application submitted to the bank made by an individual with data split across 32 columns. The fraud_bool variable indicates whether the application was fraudulent or not. The remaining variables can be segmented into two groups, those pertaining to a common group of applications the individual is a part of and those pertaining to the specific application only. The latter group can subsequently be broken into two additional segments, one containing application insights most applicants may not realize are being recorded while the other relates to personal and financial information of the applicant.

The first large segment contains attributes counting the number of applications that share a common group within a specific time frame. For example, velocity_6h keeps tracks of the average number of applications per hour within the past six hours. A similar metric, bank_branch_count_8w, tracks the number of total applications at the branch tied to the application over the past eight weeks.

Table 1. Attributes where values shared by multiple applications

Field Name	Type	Description	Range
zip_count_4w	Discrete	Number of applications within same zip code in last 4 weeks.	[1, 6700]
velocity_6h	Continuous	Velocity of total applications made in last 6 hours i.e., average number of applications per hour in the last 6 hours.	[−171, 16716]
velocity_24h	Continuous	Velocity of total applications made in last 24 hours i.e., average number of applications per hour in the last 24 hours.	[1300, 9507]
velocity_4w	Continuous	Velocity of total applications made in last 4 weeks, i.e., average number of applications per hour in the last 4 weeks.	[2825, 7000]
bank_branch_count_8w	Discrete	Number of total applications in the selected bank branch in last 8 weeks.	[0, 2385]
date_of_birth_distinct_emails_4w	Discrete	Number of emails for applicants with same date of birth in last 4 weeks.	[0, 39]

The application insight subsegment contains information about the application’s creation. Example variables include whether the application was completed through the bank’s website or app, if the application comes from a country other than that of the bank's, and the number of fraudulent applications that have been made on the device. Other metrics track the authenticity of the applicant via their inputs, such as a binary variable indicating the validity of phone numbers, whether the email address used is from a free domain, and a similarity measure between the email used and the applicant's name.

Table 2. Application insights attributes

Field Name	Type	Description	Range
keep_alive_session	Categorical (binary)	If the user ends their session via logout.	[0, 1]
device_fraud_count	Discrete	Number of fraudulent applications with used device	Was all 0.
device_distinct_emails_8w	Discrete	Number of distinct emails in banking website from the used device in last 8 weeks	[-1, 2]
name_email_similarity	Continuous	Metric of similarity between email and applicant's name.	[0, 1]
prev_address_months_count	Discrete	Number of months in previous registered address of the applicant, i.e. the applicant's previous residence, if applicable.	[-1, 383] (-1 replaced 712k missing values)
current_address_months_count	Discrete	Months in currently registered address of the applicant	[-1, 428] (-1 replaced 4254 missing values)
email_is_free	Categorical (binary)	Domain of application email (free or paid)	[0, 1]
phone_home_valid	Categorical (binary)	Validity of provided home phone.	[0, 1]
phone_mobile_valid	Categorical (binary)	Validity of provided mobile phone.	[0, 1]
session_length_in_minutes	Continuous	Length of user session in banking website in minutes	[-1, 86]
device_os	Categorical	Operative system of device that made request.	Windows, macOS, Linux, X11, or other
source	Categorical	Online source of application.	Browser (INTERNET) or app (TELEAPP).
foreign_request	Categorical (binary)	If origin country of request is different from bank's country	[0, 1]

The remaining segment has information on the applicant's age, income, housing status, employment status, housing status, and the month of the application. Financial information, such as credit score, credit limited desired, and whether the applicant has existing cards with the bank, are also recorded.

Table 3: Personal information attributes

Field Name	Type	Description	Range
income	Categorical (Brackets)	Annual income of the applicant (in decile form).	[0.1, 0.9]
credit_risk_score	Discrete	Internal score of application risk.	[-170, 389]
proposed_credit_limit	Discrete (Whole dollars)	Applicant's proposed credit limit.	[190, 2100]
customer_age	Categorical (Age bins)	Applicant's age in years, rounded to the decade.	[10, 90]
intended_balcon_amount	Continuous	Initial transferred amount for application.	[-16, 113]
employment_status	Categorical	Employment status of the applicant. Anonymized values.	"CA", "CB", "CC", "CD", "CE", "CF", "CG"
housing_status	Categorical	Current residential status for applicant. Anonymized values.	"BA", "BB", "BC", "BD", "BE", "BF", "BG"
month	Categorical	Month where the application was made.	[0, 7]
has_other_cards	Categorical (binary)	If applicant has other cards from the same banking company	[0, 1]
bank_months_count	Discrete	How old is previous account (if held) in months.	[-1, 32] (-1 replaced 253k missing values)
payment_type	Categorical	Credit payment plan type. Anonymized values.	"AA", "AB", "AC", "AD", "AE"
days_since_request	Continuous	Number of days passed since application was done	[0, 79]

It should be noted that most features have already undergone feature-engineering as the records were generated. For example, the income variable ranges from 0.1 to 0.9 as different brackets. However, a couple numerical features are not modified and maintain their unit of measure like years or count. Some categorical variables, such as employment status and housing status, take on anonymized values. Other variables, such as how old a previous bank account the applicant has is and how long the applicant has been attached to their current address, utilize -1 as a replacement value for missing values.

Initial Analysis Conducted:

The intent of the initial analysis was to determine if there are certain predictor variables which have strong relationships with the binary target variable, fraud_bool. In the case of categorical variables, this consisted of determining if there were noticeable disparities in the fraud rates of each variable's classes.

Data visualization was first used as a naïve approach to this analysis. Bar charts were created for categorical variables, with the value of each bar representing the average fraud rate. It was apparent that for some variables, there was a specific class with a much higher fraud rate than the others. For example, for the foreign request binary variable, applications coming from a foreign country had a fraud rate twice the rate of applications coming from the same country as the bank. Logically, this makes sense as it would be difficult to trace foreign bad actors down.

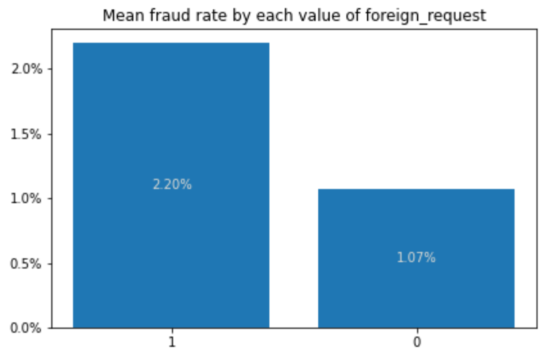


Fig. 1. Mean fraud rate by foreign_request

A similar case was with has_other_cards. Applicants who did not have an existing card with the bank were three times more likely to commit fraud. As fraudsters are trying to avoid being detected, they are probably moving from bank to bank instead of concentrating on one.

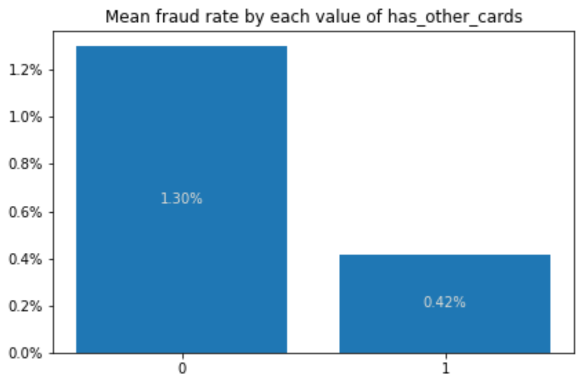


Fig. 2. Mean fraud rate by has_other_cards

A non-binary categorical variable with a large disparity in a class's fraud rate was housing_status. The "BA" housing status had a fraud rate more than four times the second highest fraud rate of the "BD" status. Not much could be interpreted from this variable to confirm this disparity, given the values were anonymized.

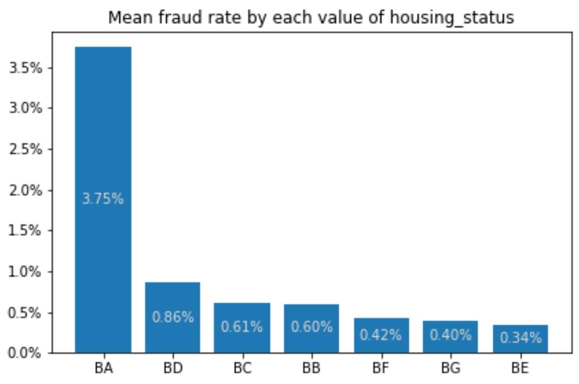


Fig. 3. Mean fraud rate by housing_status

Table 4. shows all categorical variables where there is at least a 25% difference between the categories with the lowest and highest fraud rates:

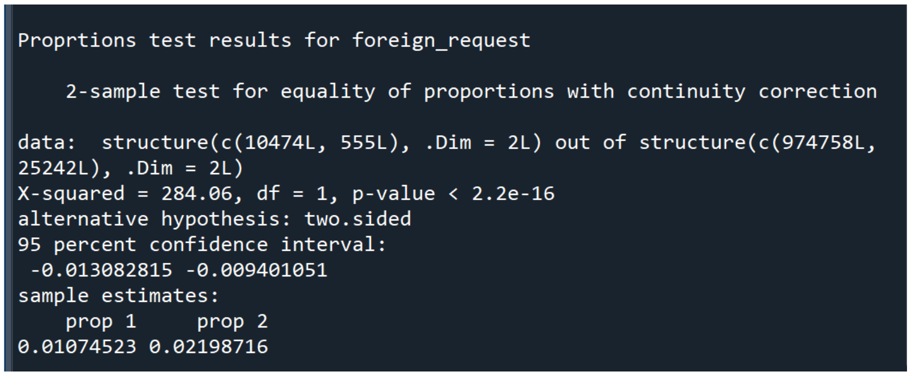
Table 4. Categorical variables with minimum 25% difference between the categories with highest and lowest fraud rates

Variable	Category with Highest Fraud Rate
device_os	Windows
email_is_free	1 (Yes)
employment_status	CC
foreign_request	1 (Yes)
keep_alive_session	0 (No)
payment_type	AC
phone_home_valid	0 (No)
phone_mobile_valid	0 (No)
income	0.9
month	7
customer_age	90
has_other_cards	0 (No)
housing_status	BA
source	Teleapp

To confirm whether the mean fraud rates of each categorical variable's classes differ from each other with statistical significance, a difference of proportions test was conducted using the prop_test function from the stats package in R (which was executed in Python). The hypotheses are as follows:

H_0: The proportion (mean fraud rate) of each group for a variable are equivalent.
H_A: The proportion of each group for a variable are not equivalent.

A sample output for the test is shown in Figure 4 for the foreign_request variable. For each categorical variable that was tested, each resulting p-value was below 0.05. Thus, at a 0.05 significance level, the null hypothesis could be rejected.



```
Proprtions test results for foreign_request

	2-sample test for equality of proportions with continuity correction

data:  structure(c(10474L, 555L), .Dim = 2L) out of structure(c(974758L,
25242L), .Dim = 2L)
X-squared = 284.06, df = 1, p-value < 2.2e-16
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.013082815 -0.009401051
sample estimates:
    prop 1     prop 2
0.01074523 0.02198716
```

Fig. 4. Difference of Proportions Test Results for foreign_request

For quantitative variables, two different visualizations were created. In both cases, data points for which -1 was imputed for missing values of certain variables were not plotted. The first was a cumulative distribution function chart. The x-axis takes on the range of values for the variable while the y-axis represents the percentage of observations less than or equal to the x-value. The aim of creating these charts was to determine if there were any noticeable differences between the distributions of the variables for each value of fraud_bool. For example, Figure 5 for name_email_similarity indicates most values where the application is not fraudulent (0) tend to be higher compared to fraudulent ones. Fraudsters may be using random emails on applications, leading to a lower similarity. The inverse is true for the credit_risk_score variable in Figure 6. A possible hypothesis for this is that fraudsters try to apply with the details of those who seem to have a better credit score to have a better chance of receiving a larger credit line.

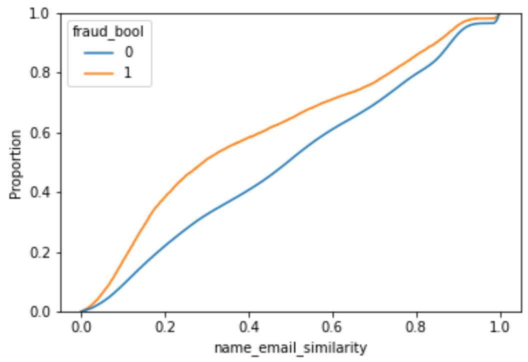


Fig. 5. Cumulative Distribution Function plot for name_email_similarity

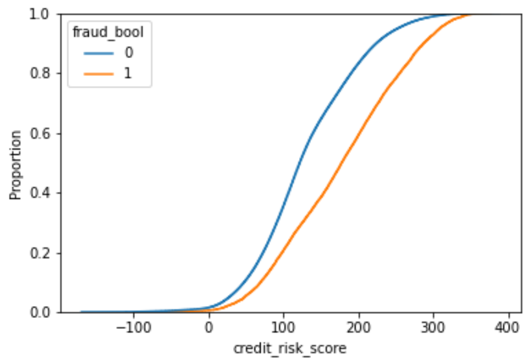


Fig. 6. Cumulative Distribution Function plot for credit_risk_score

All numerical variables where the distribution function plots were somewhat different for fraudulent and non-fraudulent applications are in Table 5 below.

Table 5. Numerical variables for which CDF plot had noticeable disparity for each value of fraud_bool

Variable	**Fraud_bool value where majority of values are larger**
Proposed_credit_limit	1
Current_address_months_count	1
Name_email_similarity	0
Credit_risk_score	1

Bar charts were also created to identify any significantly different means between each value of fraud_bool. For example, Figure 7 indicates legitimate applications had a mean intended initial transfer amount 120% more than fraudulent applications. One guess for this is that fraudsters may be unwilling to commit money upfront as part of their application. Meanwhile ,in Figure 8, fraudulent applications had a higher average requested credit limit than fraudulent ones. This is expected, as it would be reasonable to expect fraudsters to request more credit to steal.

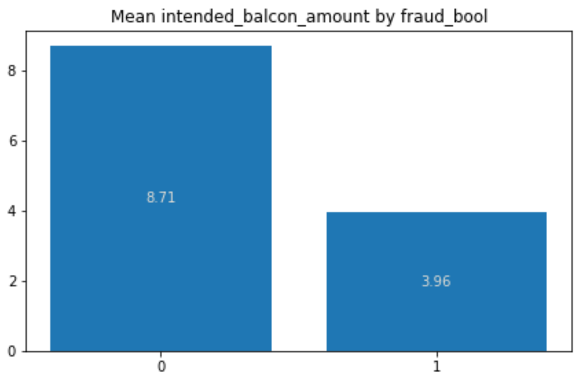


Fig. 7. Mean intended_balcon_amount by fraud_bool

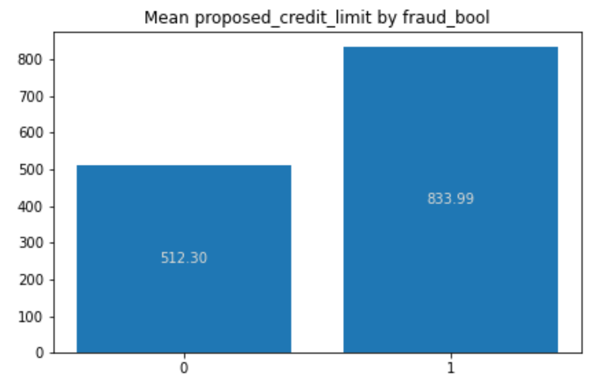


Fig. 8. Mean proposed_credit_limit by fraud_bool

The following table indicates the other variables where there is at least a 25% difference in the mean for each value of fraud_bool:

Table 6. Quantitative variables with minimum 25% difference in means for each value of fraud_bool

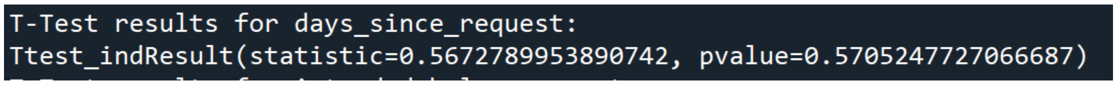
Variable	Value w/higher mean, difference
Prev_address_months_count	1, 37%
Bank_branch_count_8w	0, 38%
Credit_risk_score	1, 36%
Current_Address_months_count	1, 33%
Date_of_birth_distinct_emails_4w	0, 28%
Intended_balcon_amount	0, 120%
Name_email_similarity	0, 26%
Proposed_credit_limit	1, 63%

The final component of the preliminary analysis was a student's t-test to determine if the means for each quantitative variable, segmented by the value of fraud_bool, had a statistically significant difference or not. The hypotheses are as follows:

H_0: There is no difference in the between the means of each group (0/1 for fraud_bool).
H_A: There is a difference between the means of each group.

To perform this test, multiple conditions need to be met. The first is that the data of each group must be independent, which is true in this case. The variances of each group also must be equivalent. A simple method of checking this is calculating the variance of each group and determining if the ratio of the larger to smaller variance is less than 4. For all variables, this was true. (device_distinct_emails_8w and device_fraud_count had variances of 0 for each group; thus a null ratio was calculated.) Last was the test of normality for each group. Because each group's size was very large, this normality test can be relaxed. The normality assumption in this case would only be false if there were outliers in the data; in this case, there were none for each variable based on histograms plotted.

The t-test results returned a p-value of less than 0.05 for all variables but one, indicating that at a 0.05 significance level, the null hypothesis of the variable means for each value of fraud_bool being the same can be rejected. The lone exception was the days_since_request variable, which had a p-value of 0.5705, indicating the null hypothesis cannot be rejected.



```
T-Test results for days_since_request:
Ttest_indResult(statistic=0.5672789953890742, pvalue=0.5705247727066687)
```

Fig. 9. T-test results for days_since_request

Preliminary Findings Discussion & Next Steps:

Through the charts and statistical tests ran in the initial analysis, insight into the relationships between each variable and fraud_bool was obtained. This will help in deciding which variables should be utilized as predictors in the variables for the models that will be created. For example, days_since_request would most likely not be included as the mean value between fraudulent and non-fraudulent applications did not have a statistically significant difference. On the other hand, a categorical variable like housing_status where the bar chart indicated a noticeable difference in the fraud rate of the classes would be utilized.

Some results from the initial analysis confirmed initial assumptions about fraud characteristics, such as foreign sources and a higher proposed credit limit being more commonly tied to fraudulent applications. Others, such as a higher credit score and longer age of previous addresses on fraudulent applications were not apparent and required some inferences as to why it was the case. Whether these variables are critical to identifying fraudulent applications will depend on the models that will be created.

The next phase of the analysis will involve the creation of models that will predict whether an application is fraudulent or not. This will first involve a logistic regression using all variables to serve as a baseline performance. Following evaluation of this model, another logistic regression model will be conducted using recursive feature elimination, which will attempt to identify the most important features. This can also be conducted after removal of variables guided by the findings of the preliminary analysis and the creation of a correlation matrix to remove highly correlated variables. Other methods, such as random forest and decision tree models, will be trained and evaluated in this manner as well. The features found to have high importance values across multiple models would then be identified.

Primary Analysis:

One outstanding issue from the preliminary analysis that needed to be addressed was the presence of attributes that had an imputed value of -1 for missing values. There were three variables for which this was done: prev_address_months_count, current_address_months_count, and bank_months_count. Keeping these imputed values would have an impact on the analytical techniques that would be used, given the number of occurrences. For example, prev_address_months_count had over 712,000 records where this replacement was done. Removing all records with an imputed value not only left about 195,000 records that could be utilized in creation of predictive models but also would lower the overall fraud rate to 0.2%, approximately one-fifth the rate of the entire dataset. To maintain consistency with realistic fraud metrics, the decision was made to omit the three attributes from the analysis.

The next stage in the analysis consisted of three components: preprocessing of data, creation of baseline models using all available data, and then creation of additional models involving feature selection and hyperparameter tuning. Each models' performance would then be compared and the most important features from the top performing ones would be used to reach conclusions.

Preprocessing:

As previously mentioned, attributes of the dataset fell into qualitative or quantitative groups. For the qualitative variables, all but the income variable was considered as nominal. Income was treated as ordinal due to each value or decile being equidistant from one another. Because of this, sklearn's OrdinalEncoder was used to transform it, with each unique value receiving an integer value. The categorical variables were encoded through the creation of dummy variables with one level being dropped to serve as the baseline. These variables did need to be converted to string type first as some were initially of integer type. Meanwhile, the continuous and discrete variables were standardized using sklearn's StandardScaler object. However, the scaler was fit using only data from the training set instead of the complete set of records. According to Baeldung, the reason for doing so is that using any information from the test set during training could lead to "a potential bias in the evaluation of the performance" ("Data").

The splitting of the dataset into training and test sets was done based on the value for the month attribute of each application record. The creators of the dataset, who also trained a model on it, explain that it is typical in fraud analysis to split data temporally due to more recent data reflecting current fraud trends when a created model is put into production (Jesus et al. 6). In this case, the training set included applications from months 0-5 while the test set was made up of records from months 6 and 7. This resulted in an approximate 79.5-20.5 split of records. The month column was then removed given its use in the data split.

Evaluation Criteria:

While accuracy always remains a key consideration in evaluating the performance of models, AUC was the primary metric of focus in the context of this analysis. The low frequency of fraud cases makes the dataset heavily imbalanced; thus, accuracy alone would not be the best evaluation metric. A model predicting all applications are non-fraudulent would have an extremely high accuracy. AUC is more appropriate because of the need to consider the True Positive Rate (TPR) and False Positive Rate (FPR) in fraud classification algorithms. Sergio Jesus, one of the contributors to the dataset, mentions that while maximizing the percentage of fraud applications caught (TPR) is critical, the general practice is to also determine an appropriate FPR value as rejecting a legitimate customer's application may lead to lost business. The threshold probability value for what is deemed to be fraudulent affects the two rates, so an appropriate value is typically decided on in the real world (Jesus).

In this analysis, general model performance was more of a concern over identifying a specific best threshold. AUC is calculated based on the ROC curve, which is generated based on the TPR and FPR at different thresholds. This thus offers the desired overall performance to classify each class value correctly.

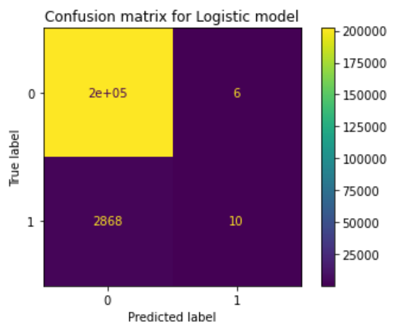
<u>Baseline Models:</u>

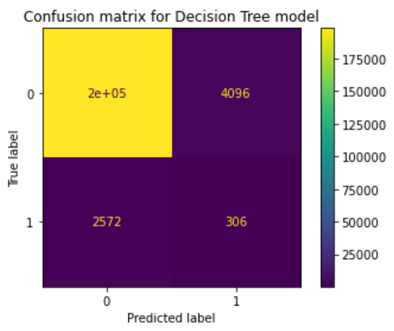
Using the training and test sets, multiple baseline models were created, including logistic regression, decision tree, and random forest. The results of each of the baseline model are displayed in the Table 7 below.

Table 7. Evaluation metrics for baseline models

Model	Training Accuracy	Test Accuracy	Test AUC
Logistic	0.99	0.99	0.88
Decision Tree	1.00	0.97	0.54
Random Forest	1.00	0.99	0.82

It should be noted these accuracies are calculated at a default threshold of 0.5 for fraudulent records. Although the accuracies of the training and test sets were very high and similar, it is not meaningful as AUC considering the problem's context. The resulting confusion matrices, Figure 10, 11, and 12, all indicate the issue in using accuracy with imbalanced classes. For each model, nearly all test set records were predicted as non-fraudulent (a slight exception can be given for the decision tree), leading to most fraudulent cases being classified incorrectly.





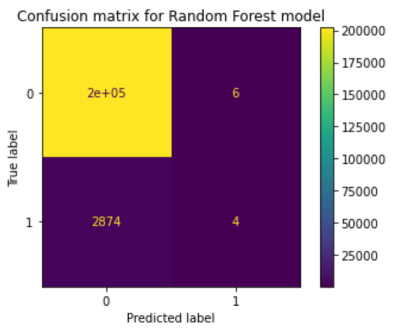
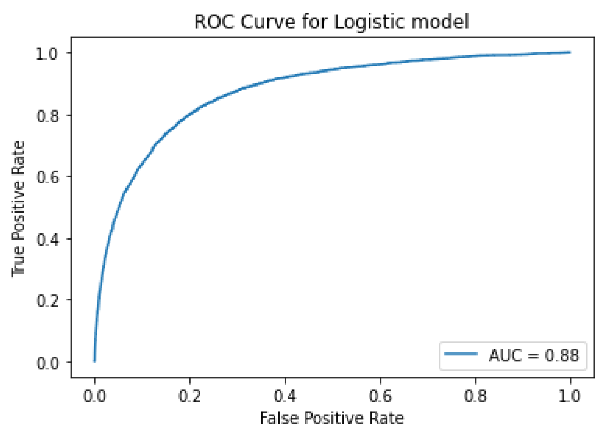
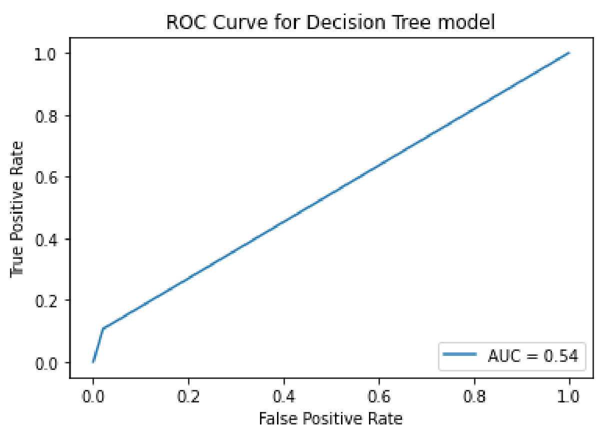


Fig. 10. (Left) Confusion matrix for logistic regression baseline model
Fig. 11. (Center) Confusion matrix for decision tree baseline model
Fig. 12. (Right) Confusion matrix for random forest baseline model

In terms of AUC, the logistic regression model performed the best while the decision tree was the worst, performing close to a random classifier as the ROC in Figure 15 is nearly a diagonal line. (AUC = 0.5)





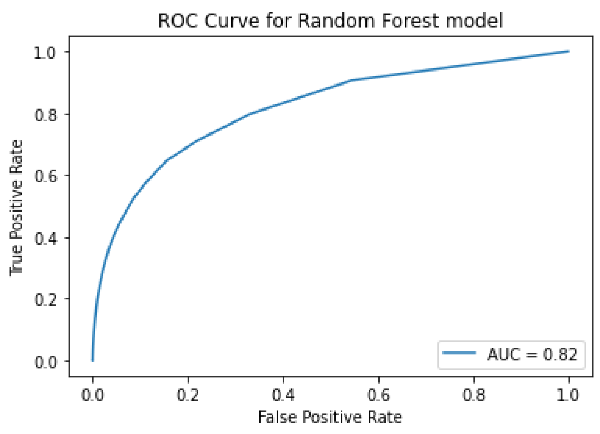


Fig. 13. (Top left) ROC Curve for logistic regression baseline model
Fig. 14. (Top right) ROC Curve for decision tree baseline model
Fig. 15. (Bottom) ROC Curve for random forest baseline model

Feature Selection & Revised Models:

With some room for improvement with the AUC metric, feature selection was done to enhance predictive capability and root out the most important variables while making model training more efficient. The original intent was to use sklearn's recursive feature elimination (RFE) to assist in feature selection. This algorithm works by training all features on a prediction algorithm, such as logistic regression, finding and eliminating the feature with the lowest importance, and repeating the process until the desired number of features remains. However, when attempting this with varying values for the number of features to be selected, the algorithm took a long time to execute. Thus, different options for feature selection were explored.

For the quantitative variables, the approach taken was to check for multicollinearity by calculating the Variance Inflation Factor, or VIF, for each variable. According to Aniruddha Bhandari, VIF is defined as "…the strength of the correlation between the independent variables. It is predicted by taking a variable and regressing it against every other variable" (Bhandari). The method described to check for multicollinearity using VIF in order to remove variables was to first calculate VIF for all variables, remove those with a value of 5 or greater, which indicates high multicollinearity, and repeat the process until VIF values were low. With the dataset's features, velocity_4w, device_distinct_emails_8w, and velocity_24h were removed in succession. Device_fraud_count was removed first as it returned a null value because of all its values being 0.

For the qualitative variables, a statistical test called Chi-squared would have been utilized to test for independence between a predictor and the target variable. Predictors found to be independent from the target would thus not be selected. This process would have involved fitting sklearn's SelectKBest algorithm to the training data using "chi2" as the scoring metric and reviewing the scores of each feature. Those with higher scores would be considered as "more relevant" and thus be selected (Brownlee).

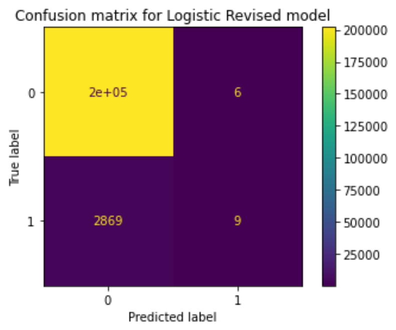
The issue with this split approach to feature selection was that it would not consider possible relationships between variables of different types. Instead, one suggestion that was made on StackExchange was to utilize Lasso regression for feature selection for linear models such as logistic regression. Meanwhile, more complex models like decision trees "should be able to learn non-linear dependencies" during which is assumed to be the process of splitting into different branches on a certain feature ("Feature"). Thus, the decision was made to utilize Lasso regression for feature selection for the revised logistic regression model and conduct hyperparameter tuning for the tree-based models to enable better selection of features.

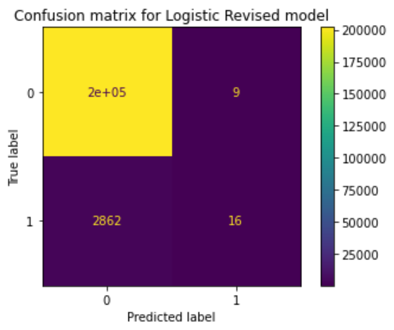
Lasso regression entails the minimization of a loss function through adjustments in the coefficient of variables, a process called regularization. Some coefficients may shrink to 0 which indicates it could be removed from the model (Sole). To perform this task, a logistic regression model was created using parameters that enable Lasso regularization and fit on the training data. One different step taken here was the ordinal encoding of all categorical variables to help identify an entire variable to eliminate, not just a specific class. This model was then wrapped in sklearn's SelectFromModel algorithm, which will provide which features are selected based on an importance threshold. In the case of Lasso regression, it will be features with a non-zero coefficient. The result was the removal of the device_fraud_count variable, which again, was most likely due to all values being 0.

A new logistic regression model was then trained on the original data, but with device_fraud_count removed. There was no difference from the baseline logistic model in the training and test accuracies, as indicated in Table 8, as well as the AUC value, identified in Figure 19. As a result, more variables were removed to see if there would be any difference. These variables included those that were found during the preliminary analysis to have less than a 25% difference in means for each value of fraud_bool: days_since_request, zip_count_4w, velocity_6h, velocity_24h, velocity_4w, and session_length_in_minutes. Again, there was no difference in evaluation metrics, except for AUC, which decreased to 0.87, as indicated in Figure 20.

The last modification that was attempted was to set the "class_weight" argument in the logistic regression model to a value of balanced. By doing so, the logistic model considers the imbalance in the target variable values and "penalizes the misclassification…of the minority class" more by giving it a larger class weight (Singh). Synthetic Minority Oversampling Technique, or SMOTE, was also another approach used to balance the frequency of classes. This technique creates additional records belonging to the minority class based off existing minority-class records. However, when compared to the results produced by the models where class weights were balanced, the models performed worse in terms of AUC. Thus, a decision was made to only use weight balancing.

The resulting evaluation metrics indicated an 80% accuracy on the training set and 79% on the test set, with an AUC value of 0.87, identified in Figure 21. Again, accuracy is not important in the problem context given it is calculated using a singular threshold (0.5 by default). However, it is apparent by the confusion matrix in Figure 18 that the balancing of class weights allowed the new logistic model to predict more fraudulent cases correctly, at the expense of increased false positives.





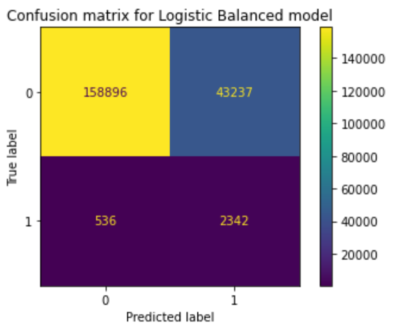
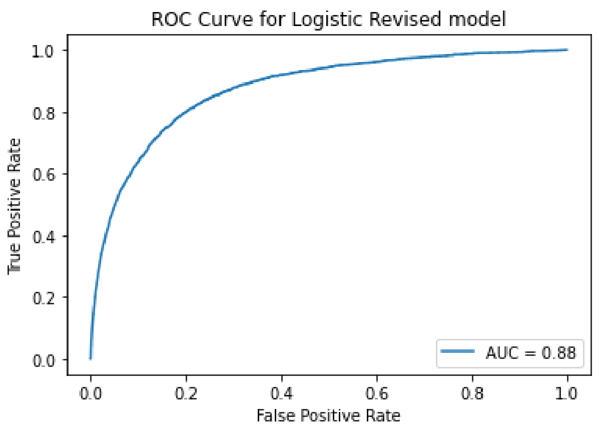
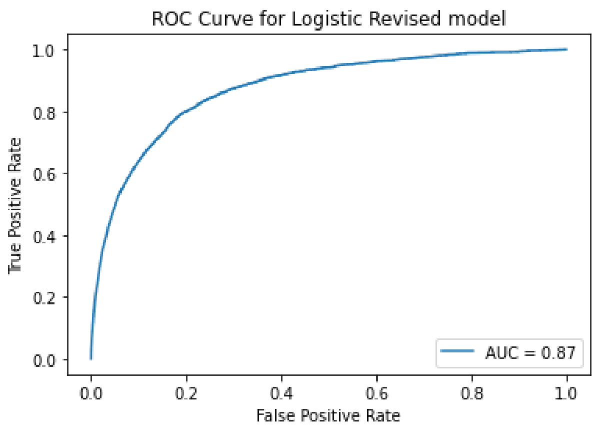


Fig. 16. (Left) Confusion matrix for revised logistic regression model
Fig. 17. (Center) Confusion matrix for revised logistic regression model, additional feature selection
Fig. 18. (Right) Confusion matrix for logistic regression model, balanced weights





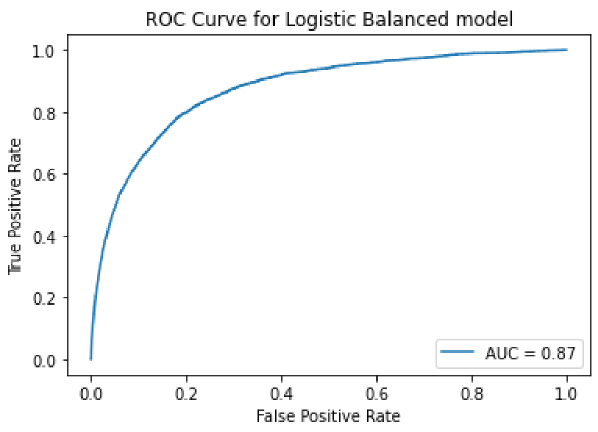


Fig. 19. (Top left) ROC Curve for revised logistic regression model
Fig. 20. (Top right) ROC Curve for revised logistic regression model, additional feature selection
Fig. 21. (Bottom) Confusion matrix for revised logistic regression model, balanced weights

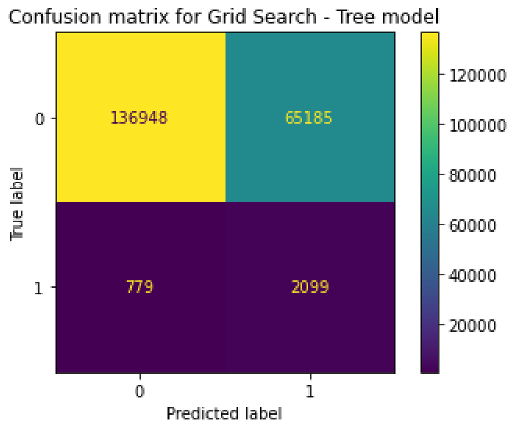
Table 8. Evaluation metrics for revised logistic models

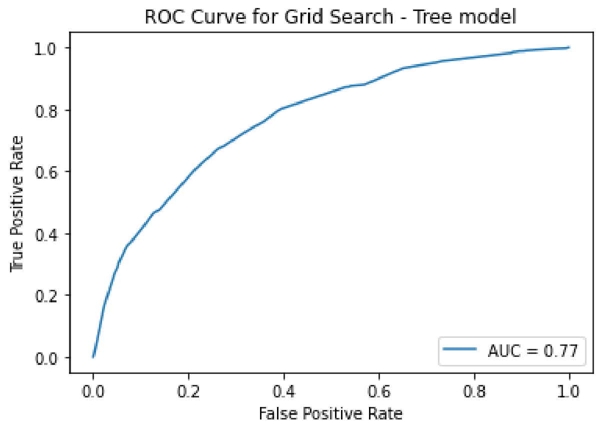
Model	**Training Accuracy**	**Test Accuracy**
Logistic Revised	0.99	0.99
Logistic Revised (additional feature selection)	0.99	0.99
Logistic Revised + Balanced Weights	0.80	0.79

Revised tree-based models were then trained using the grid-search algorithm to find the best possible hyperparameters. However, instead of creating another random forest model, an XGBoost model was created. XGBoost is like random forest in that it relies on multiple decision trees to make a prediction, however the specific usage of the trees differ. Simply put, it takes weak-performing trees and combines them to generate a stronger model (“What Is XGBoost?”). This approach was taken to obtain better model performance than what the random forest could provide. Additionally, all categorical variables for these tree-based models were ordinally encoded to help the algorithm train much faster. The reason for this is that “tree-based models…can basically learn the same information from an ordinal encoded feature as from a one-hot-encoded feature…even if the features themselves are unordered" ("Use OrdinalEncoder").

For the decision tree, multiple hyperparameters were selected for the grid search. This includes on which criteria the tree branches are split (typically entropy or Gini), the maximum depth of the tree (max_depth), the minimum number of samples a node must have to be split (min_samples_split), and the maximum number of features that are considered at each split (max_features). Selection of each hyperparameter's test values for the grid search was done using recommendations from an article by Mukesh Mithrakumar. With max_depth, the suggestion was made to identify the depth of a decision tree with no hyperparameter tuning and adjust accordingly based on training and test scores (Mithrakumar). The baseline decision tree model had a depth of 43 and given a tendency for models to overfit with greater depth as well as consideration of training time, the decision was made to use an interval from 1 to 21, in step sizes of 2. Min_sample_split used the provided recommendation of a range from 1 to 40, though a step size of 2 was also added to speed up training. The max_features hyperparameter was tested using the suggestion of "log2" due to the computational cost (in this case, time) that would be needed to train the model on over 800 thousand records. Lastly, while not discussed in the article, the class_weight parameter was set to "balanced" for the same reason as the revised logistic model.

Using "roc_auc" as the metric on which the grid search algorithm uses to select the best model, the parameters of the best decision tree had a max_depth of 7 and min_samples_split of 23. The



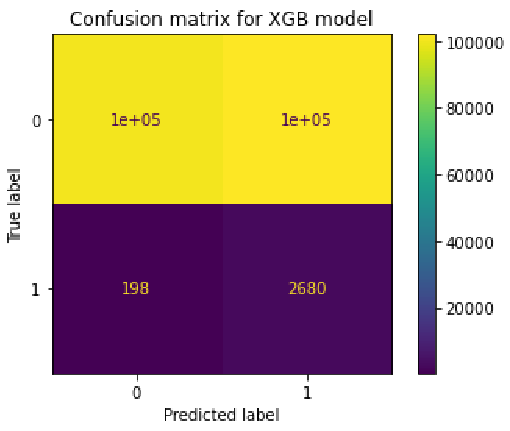


resulting ROC Curve, depicted in Figure 23, produced an AUC value of 0.77, which was worse than the baseline decision tree model.

Fig. 22. (left) Confusion matrix for decision tree model using grid search
Fig. 23. (right) ROC Curve for decision tree model using grid search

For the XGBoost model, specifically, a XGBClassifier object, a select number of hyperparameters were used in the grid search to maintain simplicity. They included the number of estimators or trees (n_estimators), the maximum depth of each tree (max_depth), the learning rate (learning_rate), which is the rate at which parameters in the model are updated, and the percentage of columns selected to be considered within each tree (colsample_bytree). The suggested ranges of values to be used for some hyperparameters were suggested in an article by Aarshay Jain. Specifically, a range of 3 to 10 was recommended for max_depth, though like before with the decision tree, a step size of 2 was utilized to minimize training time. Additionally, a range of 0.05 to 0.2 with a step size of 0.05 was used for the learning rate, although the suggested upper limit was 0.3, and 0.8, "a commonly used start value", was used for colsample_bytree, and 1 was used for scale_pos_weight (Jain). While not mentioned in the article, n_estimators is a critical component to the XGBoost models. Given the default was 100, and to not prolong training time, values of 50 and 100 were used for this hyperparameter in the grid search. Finally, to balance class weights as was done with the decision tree model, a class weight object needed to be created based off the distribution of the fraud_bool variable and passed into the sample_weight parameter on the model fit method ("XGBoost").

Using roc_auc as the scoring metric for grid search, the best model had a max_depth of 3, learning of 0.2, and 100 trees (n_estimators). The AUC of 0.86 was than the baseline random forest's AUC of 0.82. According to the confusion matrix in Figure 24, it did appear that the model was more sensitive and assigned higher probabilities to more records, resulting in over 100 thousand records being classified as false positives at threshold level of 0.5.



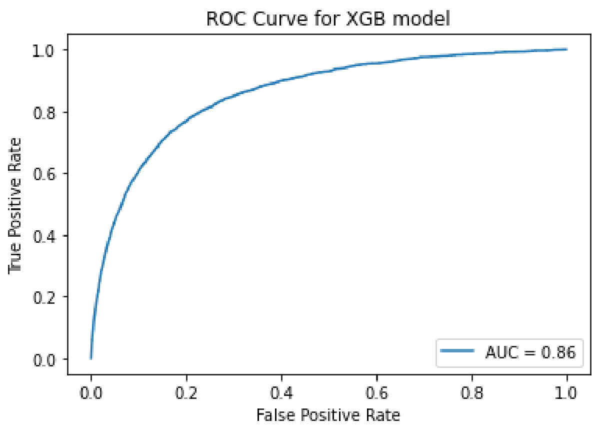


Fig. 24. (left) Confusion matrix for XGBoost model
Fig. 25. (right) ROC Curve for XGBoost model

The accuracies for each of the tree-based models using grid-search can be found in Table 9 below.

Table 9. Evaluation metrics for tree-based models

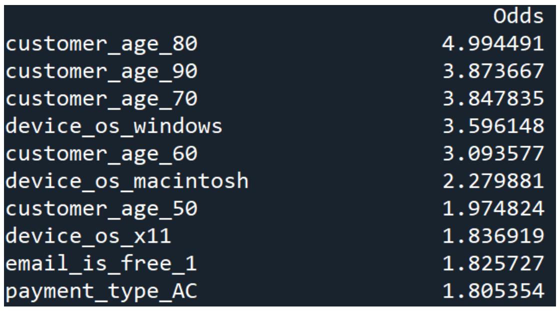
Model	Training Accuracy	Test Accuracy
Decision tree	0.81	0.68
XGBoost	0.89	0.50

Model Selection:

Looking at the best overall AUC values, both the baseline logistic and revised logistic models, without the removal of additional variables or balancing of class weights, had an AUC of 0.88. Because the latter model only had device_fraud_count removed, the focus will be on the former model. The second revised logistic model where additional variables were removed was similar, with an AUC of 0.87. On the other hand, the revised models that used balanced weights resulted in slightly lower similar values. The only difference with the revised models is that there were many false positives in the confusion matrices (threshold of 0.5) for them, specifically Figures 18, 22, and 24. Stopping the applications of too many legitimate customers is something the bank wants to avoid.

A decision was made to focus on the first two revised logistic models where there was no balancing of class weights. While the near perfect accuracies may suggest overfitting, this is again due to the imbalanced nature of the data and the default threshold of 0.5. There are not too many true positives identified in the models' confusion matrices in Figures 16 and 17. However, the high AUC value implies that a moderate decrease in the cutoff threshold used should lead to more fraudulent applications being captured and a non-significant increase in the number of false positives, with the latter leading to some decrease in the accuracies.

The next step involved looking at the most important predictors in the selected models. Because they were both logistic regression models, the coefficients can be expressed as the impact on log odds. These values were then transformed to receive the odds values. Figures 26 and 27 contain the top features for each model, by descending odds. Part of the decision to select both models despite their similarity in evaluation metrics was to identify if there were any differences in the attributes with the top odds coefficients, given the second revised logistic model had additional attributes removed. However, as both figures indicate, the top 10 were the same except for the coefficient magnitudes.


```
                                Odds
customer_age_80             4.994491
customer_age_90             3.873667
customer_age_70             3.847835
device_os_windows           3.596148
customer_age_60             3.093577
device_os_macintosh         2.279881
customer_age_50             1.974824
device_os_x11               1.836919
email_is_free_1             1.825727
payment_type_AC             1.805354
```

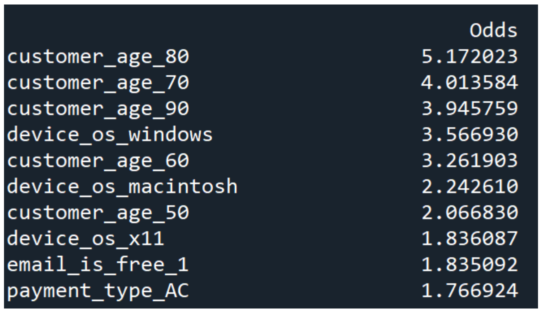

```
                                Odds
customer_age_80             5.172023
customer_age_70             4.013584
customer_age_90             3.945759
device_os_windows           3.566930
customer_age_60             3.261903
device_os_macintosh         2.242610
customer_age_50             2.066830
device_os_x11               1.836087
email_is_free_1             1.835092
payment_type_AC             1.766924
```

Fig. 26 (left). Odds coefficients for first revised logistic regression model
Fig. 27. (right) Odds coefficients for second revised logistic regression

Implications:

Although a financial institution would not directly implement one of the selected models into production for fraud detection, they can use the information from output such as Figures 26 and 27 to drive their development of a more complex model.

For example, according to Figure 26, five of the top ten odds coefficients belonged to dummy variables for the customer_age attribute, with customer_age_10 being the baseline. This confirms the trend of higher fraud rates with applications where the customer_age is in an older bin identified in Figure 28, which was created during the preliminary analysis. Logically, this would make sense as older individuals tend to be more susceptible to schemes that may compromise their identity. A bank may include rules in their model that would increasingly scrutinize the applications of older individuals by perhaps adjusting cutoff thresholds.

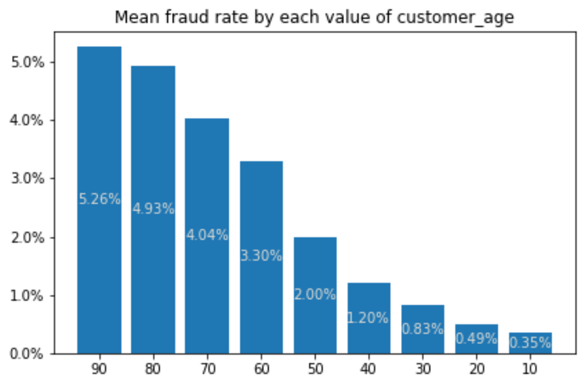


Fig. 28. Mean fraud rate of customer_age bins

Similar could be said about the email_is_free_1 dummy variable. In the preliminary analysis, it was discovered that applications using an email from a free domain had a higher average fraud rate than paid domains, as indicated in Figure 29. The bank could extend the confirmation of this insight by the logistic regression model results by isolating specific free email domains more frequently tied to fraud, and accordingly adjust tolerance levels for each one.

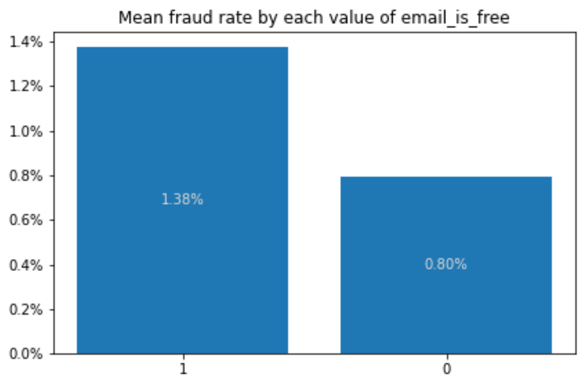


Fig. 29. Mean fraud rate of free and non-free email domains

For the device_os dummy variables, the baseline represents an OS of a type other than the three listed in Figures 26 and 27 – Windows, Macintosh, and x11. The fact that these have higher log odds compared to the "Other" group is most likely due to their prevalence in electronic devices. Although the difference between each OS's magnitude could be utilized in a similar way as the email domain example.

Essentially, the bank could repeat this investigative process for the other variables with high magnitudes, placing emphasis on those where simple descriptive statistics indicate high fraud rate disparities between classes or in the case of quantitative variables, a strong correlation with higher rates. Combined with industry knowledge of fraud trends, the bank would be able to develop models more fine-tuned to the different nuances in an applicant's submission while still also being able to generalize over a large volume of applications.

Conclusion:

Detecting fraud in the banking industry is difficult due to its highly imbalanced nature. A simple algorithm, such as a logistic regression or decision tree, is a good starting point to make these classifications but would ultimately give way to advanced models that can better generalize on enterprise-sized datasets. There must also be a delicate balance between identifying as many fraudulent cases as possible to reduce losses while not inconveniencing legitimate customers. Due to all of these considerations, the input of multiple stakeholders across the organization, from data scientists to finance leaders, is required to devise, construct, and implement a fraud detection model that meets today's evolving fraud trends.

Works Cited

Bhandari, Aniruddha. "Multicollinearity: Causes, Effects and Detection Using VIF (Updated 2023)." *Analytics Vidhya*, Analytics Vidhya, 14 Mar. 2023, www.analyticsvidhya.com/blog/2020/03/what-is-multicollinearity/.

Brownlee, Jason. "How to Perform Feature Selection with Categorical Data." *MachineLearningMastery.com*, Guiding Tech Media, 18 Aug. 2020, machinelearningmastery.com/feature-selection-with-categorical-data/.

"Data Normalization before or after Splitting a Data Set?" *Baeldung on Computer Science*, Baeldung, 19 Oct. 2020, www.baeldung.com/cs/data-normalization-before-after-splitting-set.

"Feature Selection for Data with Both Continuous and Categorical Features?" Data Science Stack Exchange, 2020, datascience.stackexchange.com/questions/68792/feature-selection-for-data-with-both-continuous-and-categorical-features.

Frost, Jim. "Multicollinearity in Regression Analysis: Problems, Detection, and Solutions." *Statistics By Jim*, 29 Jan. 2023, statisticsbyjim.com/regression/multicollinearity-in-regression-analysis/.

Jain, Aarshay. "XGBoost: Complete Guide to Parameter Tuning in XGBoost with Codes." *Analytics Vidhya*, Analytics Vidhya, 14 Mar. 2023, www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/.

Jesus, Sergio. “Evaluation in Bank Account Fraud Datasets.” *Kaggle*, Kaggle Inc., 25 Nov. 2022, www.kaggle.com/datasets/sgpjesus/bank-account-fraud-dataset-neurips-2022/discussion/372369?select=Base.csv.

Jesus, Sérgio, et al. “Turning the Tables: Biased, Imbalanced, Dynamic Tabular Datasets...” *OpenReview*, 6 June 2022, openreview.net/forum?id=UrAYT2QwOX8.

“Preventing Identity Fraud Comes down to Effective Use of Data.” *PYMNTS*, PYMNTS.com, 2 Nov. 2022, https://www.pymnts.com/fraud-prevention/2022/preventing-identity-fraud-comes-down-to-effective-use-of-data/.

Singh, Kamaldeep. “How to Improve Class Imbalance Using Class Weights in Machine Learning.” Analytics Vidhya, 2 Dec. 2022, www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/.

Sole. “Feature Selection with Lasso in Python.” Train in Data Blog, Train in Data, 16 Aug. 2022, www.blog.trainindata.com/lasso-feature-selection-with-python/.

“Use OrdinalEncoder Instead of OneHotEncoder with Tree-Based Models.” *YouTube*, Data School, 5 Oct. 2021, www.youtube.com/watch?v=n_x40CdPZss.

“What Is XGBoost?” *NVIDIA Data Science Glossary*, NVIDIA Corporation, www.nvidia.com/en-us/glossary/data-science/xgboost/.

“XGboost Python - Classifier Class Weight Option?” *Stack Overflow*, Stack Exchange Inc., stackoverflow.com/questions/42192227/xgboost-python-classifier-class-weight-option.