Summary Report: Capstone 1 Project
Fraud Detection in Mobile Payment Data
Springboard Data Science Career Track

Eric Cruz

Introduction **2**
Problem Statement 2
Dataset 2
Clients 3
Approach 3

Exploratory Data Analysis (EDA) **3**
Data Wrangling 3
Data Storytelling 6
Statistical Data Analysis 8

Test Parameters and Attributes **10**
Model Selection 10
Preprocessing 10
Convert “type” from 5 categorical class labels to 5 boolean variables 10
Define the binary categorical predictor y 10
Exclude unwanted variables per Data Wrangling 11
Train, Test, Split and Cross Validation 11
Feature Selection 11
Check for correlation and feature importance 11
Create “Change in Balance” feature from balances provided 12
Convert “step” feature to Day of Week, Time of Day 13
Parameter and Hyperparameter Selection 13
Sampling Techniques and Imbalanced Data 13
Dimensionality Reduction using PCA 14
Measuring Model Performance 14

Model Comparison “Out of the box” **14**
Supervised Models 16
LR - LogisticRegression() 17
KNN - KNeighborsClassifier() 17
SVM - svm.SVC() - C-Support Vector Classification 18
LSVM - svm.LinearSVC() - Linear Support Vector Classification 19
Inferential 19

GNB - GaussianNB() - Naive Bayes 19
Unsupervised 19
KMEANS - KMeans() 19
Neural Network 20
MLP - MLPClassifier() - Multi Layer Perceptron 20
Ensemble Bagging 20
RFC - RandomForestClassifier() 20
BRFC - BalancedRandomForestClassifier() 21
XGB - GradientBoostingClassifier() 21
Final Model Tuning **21**
Final Result 22
Future Work **23**
Parameter Tuning 23
Other models 23
Sampling data 23
Ensembles possible with combination of models 23

Introduction

Problem Statement

Online fraud is increasing and spreading rapidly across geographies and industries, and Mobile Payments represent a significant portion of the growth in both overall transaction and fraud rates. While attempting to detect and prevent fraud, the accuracy of the prediction models can have a significant impact on the ability to strike the right balance between true detection and false positives. The customer experience can be severely compromised by security measures enacted on the basis of a false positive. The fraud rate has a measurable impact on revenue, and new types and methods of fraud are evolving in response to successful detection and prevention efforts.

Dataset

Due to privacy concerns, there is little if any publicly available data for real transactions. **PaySim.csv** is a simulation of mobile money transactions with the objective to generate a synthetic transactional data set that can be used for research into fraud detection.

The dataset for this project is from the following link on Kaggle:

https://www.kaggle.com/ntnu-testimon/paysim1

Clients

The intended clients are financial institutions and merchants who use mobile payments, and will incorporate the findings into a Fraud Detection and Prevention program which covers various types of fraud attacks.

Approach

As the dataset is synthetic and designed for fraud detection research, there are several approaches that have been published as Capstone projects using the same data. Part of the exploratory analysis will involve replicating results made from contributors on github. For example, due to the imbalanced nature (e.g. fraud proportion is small relative to the population) of the data, under-sampling and over-sampling methods will be explored. As the data does not contain features other than amounts, balances, account types, and fraud flags, the explanatory features must be engineered by combining characteristics of the given fields. For example, a target condition is to check for accounts that wash balances down to zero. After reviewing the approaches taken by others, the project approach will be developed and updated.

Exploratory Data Analysis (EDA)

Data Wrangling

See link to the Jupyter Notebook used to explore the data using pandas and matplotlib:

https://github.com/cruzer42/Capstone-Project-1-for-Springboard/blob/master/Capstone%20Project1%20Data%20Wrangling.ipynb

There was no wrangling required with respect to missing values, as determined by the following procedure:

A) The csv file was converted to a pandas dataframe with pd.read_csv(), and examined with functions info(), head(), and describe().
B) A check for missing values was performed with pd.isnull().sum() and none were detected
C) However, zero values were examined and analyzed at a high level, as they may provide insights into the fraud detection patterns. In other words, zero balances have special meaning rather than indicating missing information.
D) A histogram for each column in the dataframe was generated to illustrate the imbalanced nature of the fraud case frequency in the data, and the time series frequency distribution.

Note the following description of each of the 11 columns, as well as observations about zero values and frequency distributions for each column as applicable:

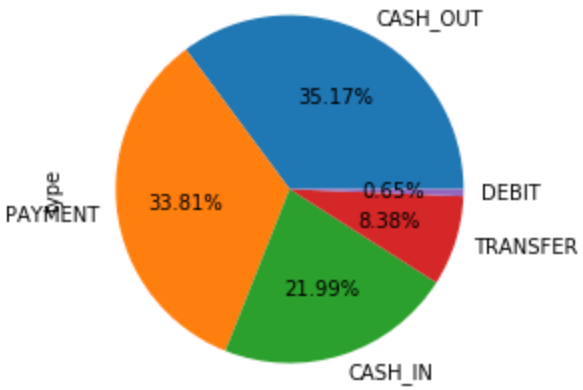
1) step - maps a unit of time in the real world. In this case 1 step is 1 hour of time. Total steps 744 (30 days simulation).
    a) Note that this is the dataset author's description posted on Kaggle, but the actual number of unique values is 743, with values 1 through 743.

b) The highest frequency is around 50,000 transactions for that hour, and there is a cluster of other intervals with a similar number of transactions. The frequency range gets as low as 2 and the distribution of frequencies appears fairly smooth.

2) type - CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.
   a) See frequency of the 5 type values:

CASH_OUT	2,237,500
PAYMENT	2,151,495
CASH_IN	1,399,284
TRANSFER	532,909
DEBIT	41,432

A pie chart was generated using matplotlib to visualize the above distribution

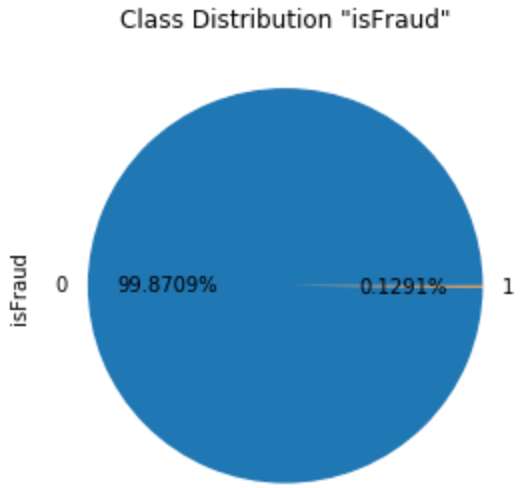


3) amount - amount of the transaction in local currency.
   a) Summary stats are included with the describe() output for all columns. The dollar amount ranges from about $92.5MM to zero, although the zero values appear to have special meaning for the 16 cases out of over 6MM observations. All are flagged as isFraud=1, so may need to be excluded or otherwise treated.
4) nameOrig - customer who started the transaction
   a) This column contains an account number, and the frequency of values was obtained using describe() on that column of the dataframe. The highest frequency for an account was 3
   b) The percentage of unique accounts is 99.8536 or 9,313 non-unique accounts out of 6,362,620
5) oldbalanceOrg - initial balance before the transaction
   a) The ratio of zero values is 0.33 or 2,102,449 of 6,362,620
6) newbalanceOrig - new balance after the transaction
   a) The ratio of zero values is 0.57 or 3,609,566 of 6,362,620
7) nameDest - customer who is the recipient of the transaction
   a) The top frequency for recipient accounts is 113, and there is a cluster of frequencies tailing off smoothly.
   b) The percentage of unique accounts is 42.7868 or 3,640,258 non-unique accounts out of 6,362,620
8) oldbalanceDest - initial balance recipient before the transaction. Note that there is not information for customers that start with M (Merchants).
   a) The ratio of zero values is 0.43 or 2,704,388 of 6,362,620

    b) Need to classify orig and dest accounts by “M” and review zero values for each

9) newbalanceDest - new balance recipient after the transaction. Note that there is not information for customers that start with M (Merchants).
    a) The ratio of zero values is 0.38 or 2,439,433 of 6,362,620
    b) Need to classify orig and dest accounts by “M” and review zero values for each

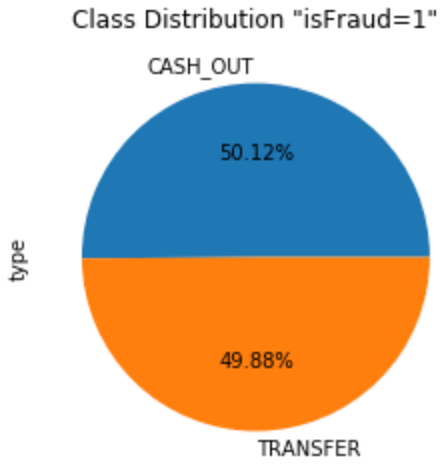
10) isFraud - This is the transactions made by the fraudulent agents inside the simulation. In this specific dataset the fraudulent behavior of the agents aims to profit by taking control or customers accounts and try to empty the funds by transferring to another account and then cashing out of the system.
    a) The ratio of zero values is 0.998709 or 6,354,407 of 6,362,620
    b) The number of isFraud cases is 8,213
    c) A pie plot was generated using matplotlib to visualize the distribution
    d) Note the distribution of isFraud=1 types is only among 2 categories, Cash Out and Transfer. These are split fairly evenly, but recall that only 8.4% of transactions are Transfer type, while 35.2% are Cash Out.
    e) Also note the 16 Amount=0, and the complete set of 16 isFlaggedFraud=1 may need special treatment or exclusion



CASH_OUT 4116

TRANSFER 4097

Name: type, dtype: int64



11) isFlaggedFraud - The business model aims to control massive transfers from one account to another and flags illegal attempts. An illegal attempt in this dataset is an attempt to transfer more than 200.000 in a single transaction.
    a) The ratio of zero values is 0.999997 or 6,362,604 of 6,362,620
    b) The number of isFlaggedFraud cases is 16, which coincidentally is the same number of transactions with Amount=0 and isFraud. Note that for each set of 16 uncommon values, they also have the feature isFraud=1. This may be a reason to exclude or otherwise treat these with special weight.
    c) All are type TRANSFER and the amounts have a pattern of matching “Orig” balances individually or in combination, and all “Dest” balances are zero.
    d) These relationships can be seen by showing head(16) on the 16 flagged values

Data Storytelling

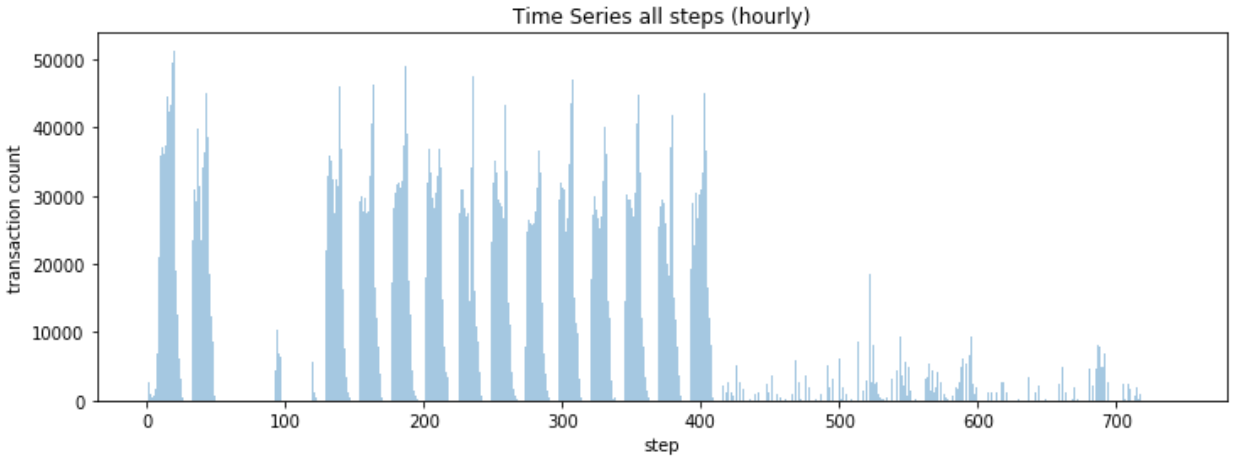
There are some data stories which emerge from Exploratory Data Analysis, which lead to a few predictions about features which could help detect fraud. See Jupyter notebook link:

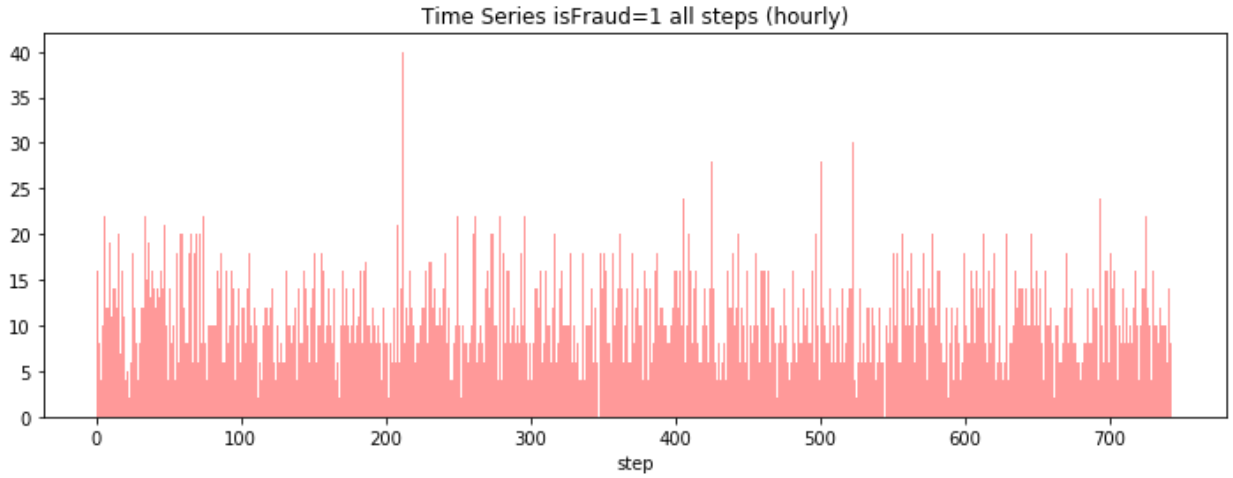
https://github.com/cruzer42/Capstone-Project-1-for-Springboard/blob/master/Capstone%20Project1%20Data%20Storytelling.ipynb

1. The timing of fraud transactions appears to be steady and consistent, contrary to other activity which contains gaps of varying sizes between active time slots.
2. The transaction “Amount” is highly correlated with the “OldBalanceOrig” for fraud transactions but not correlated for other transactions
3. The Amount on fraud transactions is in a smaller, tighter range than the distribution of amounts for all transactions.

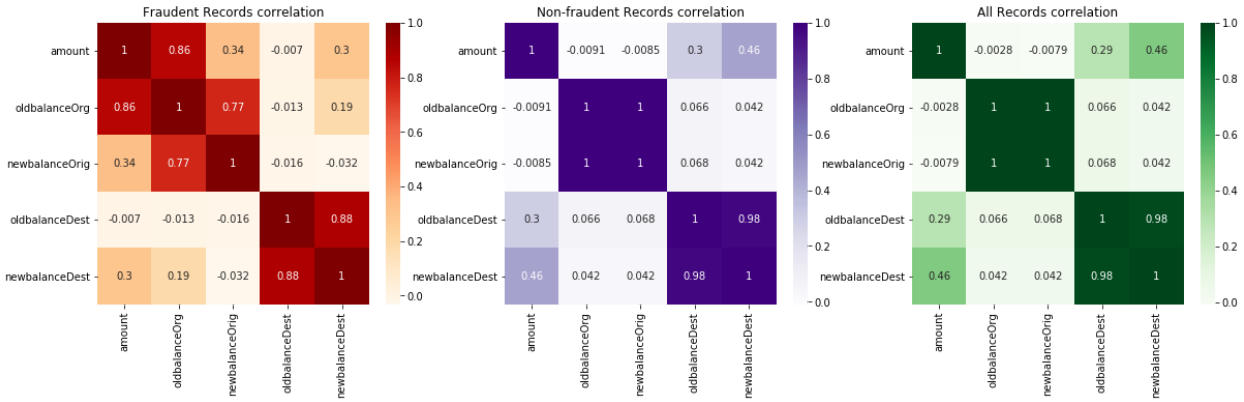
Note the following descriptions of the visualizations for the summary points above:

- The frequency distribution of transactions over time (743 steps, which is 1 hour less than 24 hours * 31 days) shows no more activity after step 718 (the final 24 hours or full last day of the data period). The frequency distribution where isFraud=1 is pretty even across the entire timeframe, which does not correspond to the non-fraud transaction distribution which contains gaps and tails off after about 400 out of 743 steps.

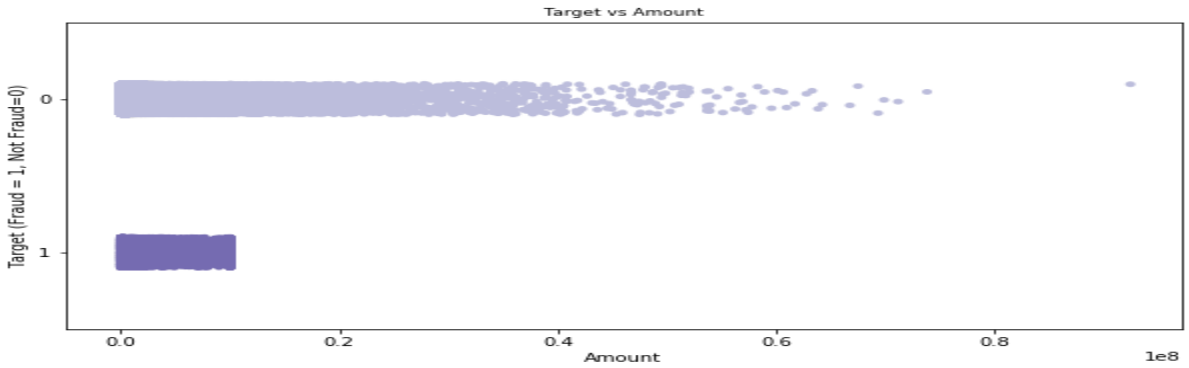




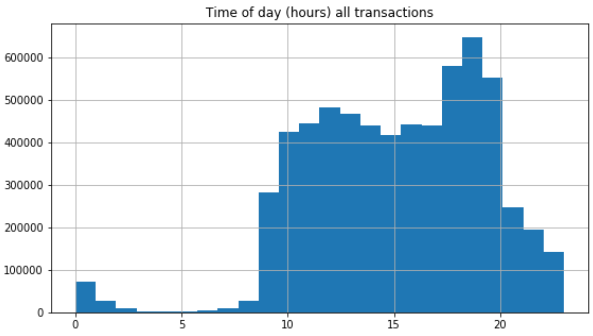
- The correlation heatmaps showing the fraud transactions, non-fraud, and combined results are plotted side-by-side. Note that fraud transactions show significantly higher correlation between “Amount” and “OldBalanceOrig” and slightly higher correlations between other features, as highlighted by the grid shading.

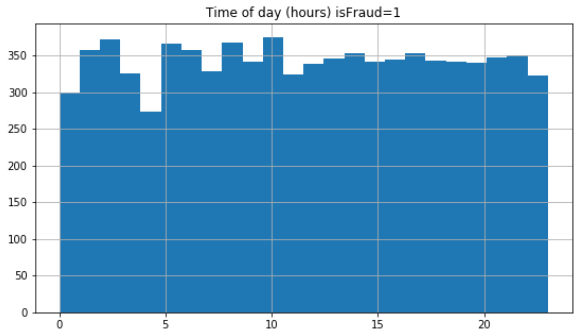


- See the stripplot of Amount for fraud vs non-fraud transactions. It suggests that amounts for fraud are generally on a smaller scale than other transactions.



- The step value was converted to date/time format, setting the first step to the default value of the first hour of 1/1/1970. Although the assumption that the step starts on a specific day or hour may be wrong, any patterns detected could easily be shifted to different starting point. It turns out that day of week does not appear to reveal a different pattern for fraud transactions, but the hourly activity of fraudulent transaction remains steady througout the period.





Statistical Data Analysis

We can perform some statistical analysis to confirm the differences which appear obvious from the visualizations.

In particular, the “Type” and “Amount” come forward in the data visualizations for reasons discussed below. See link to the statistical tests on these variable as described below:

https://github.com/cruzer42/Capstone-Project-1-for-Springboard/blob/master/Capstone%20Project1%20Statistical%20Data%20Analysis.ipynb

There is a significant subgroup in the “Type” variable because only 2 of the 5 types are represented for the target cases (e.g. isFraud=1). See the crosstab frequency table, or contingency table of Type/isFraud:

isFraud	0	1
type		
CASH_IN	1399284	0
CASH_OUT	2233384	4116
DEBIT	41432	0
PAYMENT	2151495	0
TRANSFER	528812	4097

We can perform the Pearson's Chi-squared test of the null hypothesis that the frequency of categorical variable "Type" is independent across the "isFraud" categories. As expected, the p-value is virtually zero, so the null hypothesis is rejected. This indicates dependence of the categorical variable on the category.

stats.chi2_contingency()

(22082.53571319108, 0.0, 4, array([[1.39747778e+06, 1.80622440e+03],
[2.23461179e+06, 2.88821075e+03],
[4.13785187e+04, 5.34812728e+01],
[2.14871781e+06, 2.77719374e+03],
[5.32221110e+05, 6.87889834e+02]]))

The more important subgroup is the target variable itself, the "isFraud" boolean indicator. The dataset is highly imbalanced with only 0.13%, or about 8 thousand of approximately 6 million transactions. As shown in the Data Storytelling analysis, the visualization of "Amount" shows the fraud transactions in a smaller, tighther range than the nonfraud transactions. We can test the null hypothesis that the mean "Amount" is the same for both categories, using a t-test. See the mean and std calculated by category, along with the corresponding p-value calculated from the t-test.

The number of nonfraud amounts is 6354407
The number of fraud amounts is 8213
The std amount nonfraud is 596236.9813471739
The std amount with fraud is 2404252.9472401612
The mean amount nonfraud is 178197.04172739814
The mean amount with fraud is 1467967.299140387
The calculated stdev_diff is 602079.9804398556
The mean fraud minus nonfraud amount is 1289770.257412989
The calculated t-stat is 194.01200466038233
ttest_ind(fraud_amount, nonfraud_amount)
Ttest_indResult(statistic=194.01200466037974, pvalue=0.0)

The resulting p-value of virtually zero indicates that the null hypothesis that the Amount is the same for both categories is rejected, or the Amount is statistically different.

Also shown in the Data Storytelling analysis, there is a strong correlation between "Amount" and "Old Balance Orig" which can be seen in the fraud transactions but not the non-fraud transactions, and higher but not quite as significant between "Amount" and "New Balance Orig". This is plausible given the provided description "In this specific dataset the fraudulent behavior of the agents aims to profit by taking control or customers accounts and try to empty the funds by transferring to another account and then cashing out of the system."

The Paysim dataset is a synthetic simulation of mobile payment data, because actual data is generally not available due to proprietary concerns. This means the creation of underlying data can be considered a bootstrap approach. The test to determine fraud is inferential, but not exactly along the lines of traditional hypothesis testing due to the imbalance observed in the target boolean variable. Rather than a linear regression analysis, a more appropriate approach for binary classification is logistic regression. It also seems plausible to consider a Bayesian approach, such as the GaussianNB model. However, there are several other models that may be appropriate given the binary classification problem, which will be considered as we learn about models in upcoming sections of the course. Some potential candidates are Random Forest Classifier, Support Vector Machine, and KNN just to name a few.

Test Parameters and Attributes

Model Selection

Since this type of fraud detection is a binary classification problem, there are several Machine Learning Models which are usually suggested as good candidates for this type of problem. As we review each candidate, we will describe some model characteristics and parameters, in order to determine whether the results confirm our intuition about its suitability for this particular problem.

Note that our data includes labels for the classification task, so we will start with Supervised Learning models such a regression (Logistic Regression), decision trees (SVM), and clustering (KNN). We will also include a Bayesian Inferential Statistics approach, or Naive Bayes, such as MultiNomialNB() and GaussianNB(). A Neural Network MLP Classifier, and Ensemble Learning methods, Random Forest Classifier and Gradient Boosting Classifier, will be tested. We will use Cross Validation and Pipelining, and test hybrid combinations to develop the final model.

Unsupervised Learning models will be tested in the ensemble pipeline. PCA is an unsupervised dimensionality reduction technique that will be used in the feature selection analysis and implementation. For the sake of comparison, a KMeans clustering model will be compared to its supervised counterpart, KNN.

Preprocessing

The data will be centered and normalized using the preprocessing function StandardScaler(). Further, we can convert the “Type” variable, for which we rejected the null hypothesis that distribution of type is independent of fraud, into a categorical feature indicating true/false for each of the 5 types. We will also create new features by combining some which are intuitively related, such as the difference between old and new balance because the fraud problem was described in the dataset description as typically emptying out an account’s entire balance. In addition to checking for correlation between all original, transformed, and created features, we will use GridsearchCV (cross validation) to measure and compare feature importance using various combinations. Those results will help determine which features to include in the final model.

Convert “type” from 5 categorical class labels to 5 boolean variables

As noted earlier, the “type” variable contains 5 values, but only CASH_OUT and TRANSFER have any activity identified as fraud. For a classification task in general, the data must be numerical, and for binary classification the integers represent a true/false label.

Define the binary categorical predictor y

The “isfraud” variable provides the classification labels, and represents the result we are trying to predict. So X is the array of numerical values we transform, create, or keep and y is the target label.

Exclude unwanted variables per Data Wrangling

The “nameDest, nameOrig” fields contain a character identifying a Customer vs Merchant account, but the merchant balances are not available. Per the data exploration, the account numbers did not reveal any significant patterns. Therefore, these fields are excluded along with the “isflaggedfraud” indicator which only applies to a miniscule number of observations which are nonsensical anyway. Note that we transform the “step” variable to a datetime value in order to extract other features, and exclude the transformed interim field because it is redundant and incompatible.

Train, Test, Split and Cross Validation

Per standard practice, we will use the Train, Test, Split utility in sklearn in order to partition the dataset. This helps address the problem of overfitting, or learning the training data well without being able to generalize to the unseen test data, and underfitting which is like not performing well even on the training data. The default split is 25%, but we will use 30% which is a commonly used value. Cross Validation will also be employed using GridSearchCV with default parameters. The default method is StratifiedKFold, a variation of KFold that returns stratified folds. The folds are made by preserving the percentage of samples for each class. In KFold Cross Validation we split our data into k different subsets (or folds). We use k-1 subsets to train our data and leave the last subset (or the last fold) as test data. We then average the model against each of the folds and then finalize our model. After that we test it against the test set.

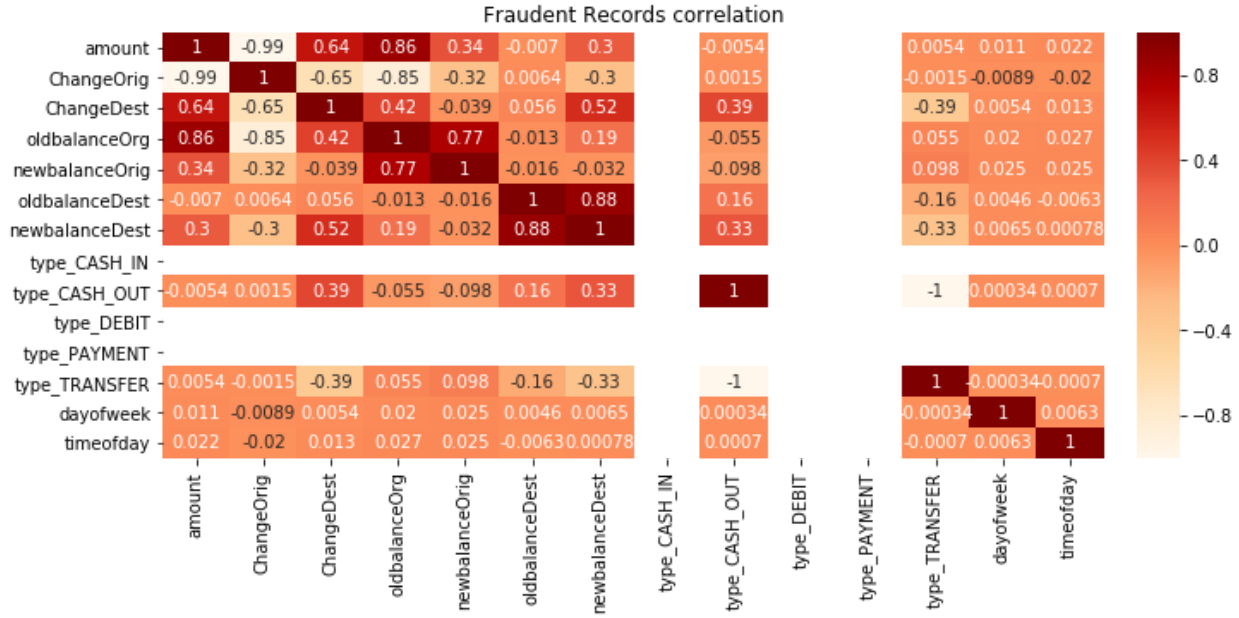
Feature Selection

As this is a binary classification problem, we are not as interested in finding the most important features as we might be for a multiple class problem. Therefore, we are not as concerned with correlation between variables including those we construct from other features. Nevertheless, it is interesting to review how the feature importance shifts as we include the additional features with improved test results. As the Random Forest Classifier appears to be the best model in the baseline comparison, we use it to determine which features to include in our final model.

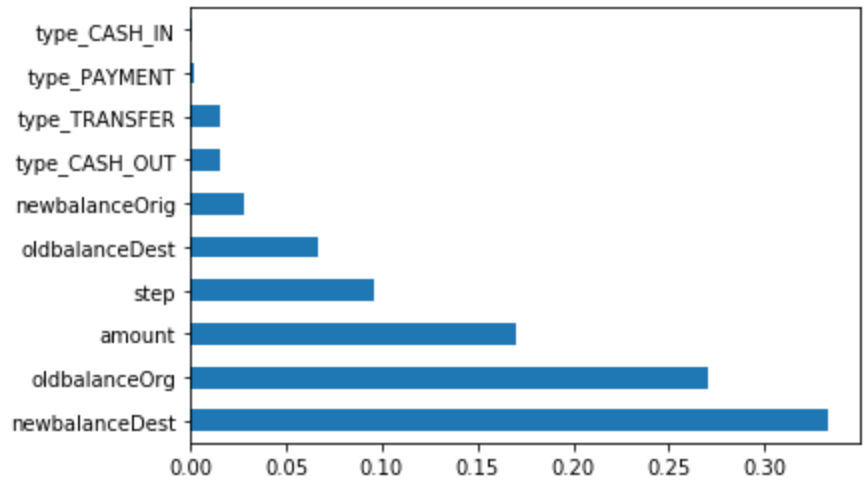
https://github.com/cruzer42/Capstone-Project-1-for-Springboard/blob/master/Capstone%20Project%201-rfc-feature_selection-defaultparams.ipynb

Check for correlation and feature importance

The matrix with all the features selected for the final model is shown below.
The most highly correlated feature pairs are **Old Balance Dest with New Balance Dest (0.88)** and **Amount with OldBalanceOrig(0.86)**, and **Amount with ChangeOrig (-0.99), and Old Balance Orig with ChangeOrig (-.85)** show the most negative correlation. The following plot was generated with sns heatmap.

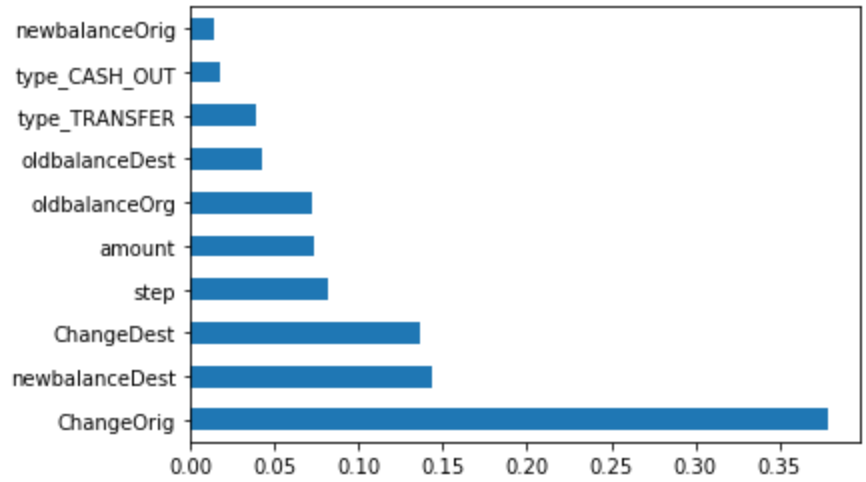


We can use the feature importance attribute of the RFC ensemble model in sklearn, and visualize the results. Note the correlated features are also among the highest scoring.



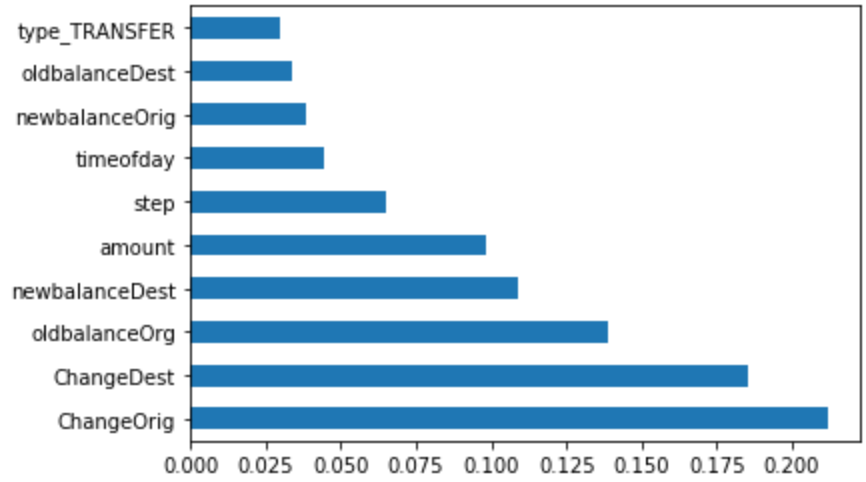
Create “Change in Balance” feature from balances provided

Note that the Change features dominate the top few spots for feature importance.



Convert “step” feature to Day of Week, Time of Day

When adding the features for Day of Week and Time of Day, neither feature ranks near the top but the distribution of measured importance is distributed more evenly when including the new features. Note the decision to keep the additional features is based on observing improvement in model performance per the metrics and confusion matrix, despite high correlation between features.



See the improvement in metrics corresponding adding the feature sets above :

Confusion Matrix	**[[1906300 51]**	**[[1906288 63]**	**[[1906322 29]**
	[546 1889]]	**[487 1948]]**	**[445 1990]]**
Model	**start features**	**plus Change balance**	**plus Time conversion**
Accuracy score	**0.9997**	**0.9997**	**0.9998**
F1 score	**0.8635**	**0.8763**	**0.8936**
Precision score	**0.9737**	**0.9687**	**0.9856**
Recall score	**0.7758**	**0.8000**	**0.8172**
ROC-AUC score	**0.8879**	**0.9000**	**0.9086**

Parameter and Hyperparameter Selection

The selection of parameters, and review of hyperparameters will be described for each model in the presentation of results, as we will iteratively review the tradeoffs between performance metrics and their applicability to our business goals.

Sampling Techniques and Imbalanced Data

A popular suggestion for dealing with the type of class imbalance problem observed in this dataset is SMOTE, or Synthetic Minority Oversampling Technique. The basic idea behind this technique is to adjust for the imbalance by synthetically balancing out the samples between the

2 classes. In addition, some of the models have a parameter for “class weight”, which is similar in intent when choosing the value ‘balanced’. Although it sounds intuitively promising, using these methods did not help as much as expected, and diminished performance in many cases.

Dimensionality Reduction using PCA

Another technique which can benefit classification problems is dimensionality reduction using PCA. This would apply more so to a problem with multiple classes and many features, and might also be used to analyze feature selection. Another reason to test this technique is to observe performance in terms of run time, or memory consumed. For example, if PCA speeds up a slow model significantly, it could be used to obtain results for parameter tuning that otherwise would not be obtainable. The overall results across the models tested were not an improvement, and combining PCA with SMOTE at the same time did not lead to any significant improvements.

Measuring Model Performance

The results will be evaluated using metrics from the model fitting scores. We will unpack the confusion matrix, comparing the classification report and score function results to clarify the ordering of the matrix labels. We will plot the ROC curve, and also review the Precision, Accuracy, Recall, and F1 scores. Interpreting the metrics will be subject to our business goals, and the relative tradeoffs can be measured by this output.

The metrics selected for review are from sklearn.metrics: Accuracy, Precision, Recall, F1, and ROC-AUC. These measures are related to the Confusion Matrix, which shows the classification of results in 4 categories: True Positive, True Negative, False Positive, False Negative. When reviewing the results among the 10 models and 5 metrics shown, we can see how using a single measure can be misleading. I find it helpful to review the Confusion Matrix, as the visualization of the diagonals makes it easier to see the False Positive/False Negative tradeoff in my opinion.

Model Comparison “Out of the box”

The models were selected with a goal of covering all the types presented thus far in the curriculum at the Capstone Project 1 stage. During the course of exploring parameters, 2 additional models were added as an extension of models in the original group. The Random Forest Classifier **RFC** stands apart with the best results, but a few other models perform well enough to be considered for tuning. A few of the models perform very poorly, but part of the reason for including them anyway is to compare run time and learn about available parameters. There is a limitation for some models in terms of prohibitive run times which impacts the decision on whether it is expedient to tune parameters. Note run time in the 5 hour range for KNeighbors Classifier **KNN** and Support Vector Model **SVM,** compared to others which take less than a minute on the same scale.

The test results were obtained using a pre-processed version of the database, which includes the new features in order to avoid the time consuming step of performing the time conversion again. Each model is run in a pipeline to include the Standard Scaler function to normalize the data. Note that GridsearchCV, or cross validation is not used in this test because it multiplies

the runtime by the number of folds, which defaults to 5 with a minimum of 2. When we tune the best model and measure performance, we will pursue the most reliable results using cross validation.

Default	Supervised				Inferential	Unsupervised	Neural Network	Ensemble Bagging		
Confusion Matrix	[[1906208 143] [1183 1252]]	[[1906292 59] [905 1530]]	[[1906326 25] [998 1437]]	[[1906306 45] [1482 953]]	[[1078399 827952] [0 2435]]	[[1485870 420481] [2433 2]]	[[1906248 103] [529 1906]]	[[1906325 26] [439 1996]]	[[1886432 19919] [19 2416]]	[[1905852 499] [1988 447]]
Model	lr	knn	svm	lsvm	gnb	kmeans	mlp	rfc	brfc	xgb
Accuracy score	0.9993	0.9995	0.9995	0.9992	0.5662	0.7784	0.9997	0.9998	0.9896	0.9987
F1 score	0.6538	0.7604	0.7375	0.5552	0.0058	0.0000	0.8578	0.8957	0.1951	0.2644
Precision score	0.8975	0.9629	0.9829	0.9549	0.0029	0.0000	0.9487	0.9871	0.1082	0.4725
Recall score	0.5142	0.6283	0.5901	0.3914	1.0000	0.0008	0.7828	0.8197	0.9922	0.1836
ROC-AUC score	0.7570	0.8142	0.7951		0.7828	0.3901	0.8913	0.9098	0.9909	0.5917
Wall time	Wall time: 51.9 s	Wall time: 5h 35min 2s	Wall time: 4h 26min 49s	Wall time: 14min 22s	Wall time: 14.5 s	Wall time: 55.6 s	Wall time: 15min 54s	Wall time: 7min	Wall time: 4min 12s	Wall time: 20min 57s

https://github.com/cruzer42/Capstone-Project-1-for-Springboard/blob/master/Capstone%20Project%201-loadfeatures-default_params.ipynb

The results while including PCA were generally equal or worse in terms of performance measures, but **KNN run time was reduced from 5+ hours to 3 minutes**. The run time results vary from significant reductions to substantial increases.

PCA	Supervised				Inferential	Unsupervised	Neural Network	Ensemble Bagging		
Confusion Matrix	[[1906208 143] [1183 1252]]	[[1906278 73] [891 1544]]	[[1906326 25] [998 1437]]	[[1906305 46] [1482 953]]	[[1872416 33935] [1396 1039]]	[[420476 1485875] [2 2433]]	[[1906271 80] [633 1802]]	[[1906300 51] [751 1684]]	[[1843533 62818] [80 2355]]	[[1906278 73] [2275 160]]
Model	lr	knn	svm	lsvm	gnb	kmeans	mlp	rfc	brfc	xgb
Accuracy score	0.9993	0.9995	0.9995	0.9992	0.9815	0.2216	0.9996	0.9996	0.9670	0.9988
F1 score	0.6538	0.7621	0.7375	0.5550	0.0555	0.0033	0.8348	0.8077	0.0697	0.1199
Precision score	0.8975	0.9549	0.9829	0.9540	0.0297	0.0016	0.9575	0.9706	0.0361	0.6867
Recall score	0.5142	0.6341	0.5901	0.3914	0.4267	0.9992	0.7400	0.6916	0.9671	0.0657
ROC-AUC score	0.7570	0.8170	0.7951		0.7044	0.6099	0.8700	0.8458	0.9671	0.5328
Wall time	Wall time: 59.3 s	Wall time: 3min 14s	Wall time: 5h 11min 3s	Wall time: 13min 59s	Wall time: 17.4 s	Wall time: 1min 5s	Wall time: 8min 6s	Wall time: 7min 22s	Wall time: 4min 30s	Wall time: 32min 6s

https://github.com/cruzer42/Capstone-Project-1-for-Springboard/blob/master/Capstone%20Project%201-loadfeatures-default_params-PCA.ipynb

The results while including SMOTE, but not PCA, are notably worse for all models with the exception of **KMEANS,** with a dramatic reduction in False Positives. The initial result appeared not worth pursuing, but with SMOTE it seems possible that further tuning could produce good results. Note the **KNN** run time increases from the 5 hour range to 18+ hours. Regarding the **SVM** run time, it was longer than 18 hours and the process was aborted. As an alternative, parameter tweaking led to checking the kernel 'linear' setting, and a corresponding review of linear version **LSVM** model.

SMOTE	Supervised		*not SMOTE, rbf		Inferential	Unsupervised	Neural Network	Ensemble Bagging		
Confusion Matrix	[[1823974 82377] [128 2307]]	[[1902026 4325] [456 1979]]	[[21860 1884491] [8 2427]]	[[1820376 85975] [153 2282]]	[[1081677 824674] [0 2435]]	[[1906342 9] [2180 255]]	[[1901094 5257] [52 2383]]	[[1905515 836] [144 2291]]	[[1905338 1013] [97 2338]]	[[1881366 24985] [44 2391]]
Model	lr	knn	svm(linear, max1k)	lsvm	gnb	kmeans	mlp	rfc	brfc	xgb
Accuracy score	0.95678	0.99750	0.01272	0.95488	0.56796	0.99885	0.99722	0.99949	0.99942	0.98689
F1 score	0.05296	0.45291	0.00257	0.05032	0.00587	0.18896	0.47305	0.82380	0.80816	0.16041
Precision score	0.02724	0.31393	0.00129	0.02586	0.00294	0.96591	0.31191	0.73265	0.69770	0.08734
Recall score	0.94743	0.81273	0.99671	0.93717	1.00000	0.10472	0.97864	0.94086	0.96016	0.98193
ROC-AUC score	0.95211	0.90523			0.78370	0.55236	0.98794	0.97021	0.97982	0.98441
Wall time	Wall time: 2min 13s	Wall time: 18h 34min 20s	Wall time: 3min 21s	Wall time: 33min 38s	Wall time: 39.6 s	Wall time: 2min 10s	Wall time: 3h 43min 30s	Wall time: 5min 42s	Wall time: 1h 1min 53s	Wall time: 41min 48s

https://github.com/cruzer42/Capstone-Project-1-for-Springboard/blob/master/Capstone%20Project%201-loadfeatures-default_params-smote.ipynb

The results for models with a “class_weight” feature that could be set to ‘balanced’ are shown here, with substitution of linear for SVC. The the RFC model result with ‘balance’ equals the best obtained with tuned parameters, but all other results were notably worse, including when combining both class_weight=’balanced’ with SMOTE.

Balanced	Supervised				Inferential	Unsupervised	Neural Network	Ensemble Bagging	
Confusion Matrix	[[1819643 86708] [140 2295]]			[[1894472 11879] [397 2038]]				[[1906332 19] [482 1953]]	[[1877057 29294] [20 2415]]
Model	lr	knn	svm	lsvm	gnb	kmeans	mlp	rfc	brfc
Accuracy score	0.9545			0.9936				0.9997	0.9846
F1 score	0.0502			0.2493				0.8863	0.1415
Precision score	0.0258			0.1464				0.9904	0.0762
Recall score	0.9425			0.8370				0.8021	0.9918
ROC-AUC score	0.9485			0.9154				0.9010	0.9882
Wall time	Wall time: 1min 36s			Wall time: 21min				Wall time: 2min 27s	Wall time: 4min 10s

Balanced and SMOTE	Supervised				Inferential	Unsupervised	Neural Network	Ensemble Bagging	
Confusion Matrix	[[1823668 82683] [126 2309]]			[[1820814 85537] [154 2281]]				[[1905545 806] [162 2273]]	[[1905300 1051] [93 2342]]
Model	lr	knn	svm	lsvm	gnb	kmeans	mlp	rfc	brfc
Accuracy score	0.9566			0.9551				0.9995	0.9994
F1 score	0.0528			0.0505				0.8244	0.8037
Precision score	0.0272			0.0260				0.7382	0.6902
Recall score	0.9483			0.9368				0.9335	0.9618
ROC-AUC score	0.9524			0.9459				0.9665	0.9806
Wall time	Wall time: 2min			Wall time: 38min 13s				Wall time: 8min 1s	Wall time: 3h 13min 14s

https://github.com/cruzer42/Capstone-Project-1-for-Springboard/blob/master/Capstone%20Project%201-loadfeatures-balanced.ipynb

https://github.com/cruzer42/Capstone-Project-1-for-Springboard/blob/master/Capstone%20Project%201-loadfeatures-balanced-smote.ipynb

As a sanity check, the results were generated where possible combining both SMOTE and PCA with Standard Scaler in the pipeline. There was no meaningful improvement in metrics or runtime across the models in general, and the result for the **RFC** preferred model is the worst out of all the combinations in the test.

PCA and SMOTE	Supervised				Inferential	Unsupervised	Neural Network	Ensemble Bagging		
Confusion Matrix	[[1823999 82352] [128 2307]]	[[1902025 4326] [456 1979]]		[[1820744 85607] [154 2281]]	[[1862508 43843] [1247 1188]]	[[1906342 9] [2180 255]]	[[1899103 7248] [43 2392]]	[[1904884 1467] [418 2017]]	[[1904689 1662] [382 2053]]	[[1841299 65052] [91 2344]]
Model	lr	knn	svm	lsvm	gnb	kmeans	mlp	rfc	brfc	xgb
Accuracy score	0.9568	0.9975		0.9551	0.9764	0.9989	0.9962	0.9990	0.9989	0.9659
F1 score	0.0530	0.4529		0.0505	0.0501	0.1890	0.3962	0.6815	0.6676	0.0671
Precision score	0.0273	0.3139		0.0260	0.0264	0.9659	0.2481	0.5789	0.5526	0.0348
Recall score	0.9474	0.8127		0.9368	0.4879	0.1047	0.9823	0.8283	0.8431	0.9626
ROC-AUC score	0.9521	0.9052			0.7324	0.5524	0.9893	0.9138	0.9211	0.9643
Wall time	Wall time: 1min 55s	Wall time: 5min 45s		Wall time: 34min 34s	Wall time: 47.4 s	Wall time: 2min 14s	Wall time: 3h 26min 16s	Wall time: 21min 33s	Wall time: 5h 54min 9s	Wall time: 1h 25min 33s

https://github.com/cruzer42/Capstone-Project-1-for-Springboard/blob/master/Capstone%20Project%201-loadfeatures-default_params-smote-pca.ipynb

Supervised Models

LR - LogisticRegression()

The Logistic Regression model is often mentioned as a good candidate for binary classification. It is similar to the Linear Regression model but predicts an output of true or false.
Several other models performed better by certain measures, but the model runs pretty fast making it possible to try tuning the model parameters.

- Parameter tuning
    - Regularization parameter default is c=1.0. Smaller values represent greater regularization so the value functions as an inverse. A large range of values was tested, and the Gridsearch result selected a value of 1000. However, the incremental gains are minimal.
    - The maximum iteration to converge default is max_iter=100. A warning was given that a future version will increase to 1000, so a few values were tested. It seems intuitive that using too high a number could increase run time without necessarily gaining incremental benefits to justify the overhead.
    - The default model will be defined as solver='lbfgs' per a future warning message. There was a negligible improvement in the F1/ROC-AUC measures, and runtime increased in proportion to the 5-fold validation increase in number of runs.

Confusion Matrix	[[1906208 143] [1183 1252]]	[[1906202 149] [1165 1270]]
Model	lr default	lr best_params
Accuracy score	0.9993	0.9993
F1 score	0.6538	0.6591
Precision score	0.8975	0.8950
Recall score	0.5142	0.5216
ROC-AUC score	0.7570	0.7607
Wall time	Wall time: 59.3 s	Wall time: 5min 47s

https://github.com/cruzer42/Capstone-Project-1-for-Springboard/blob/master/Capstone%20Project%201-loadfeatures-lr-gridsearch.ipynb

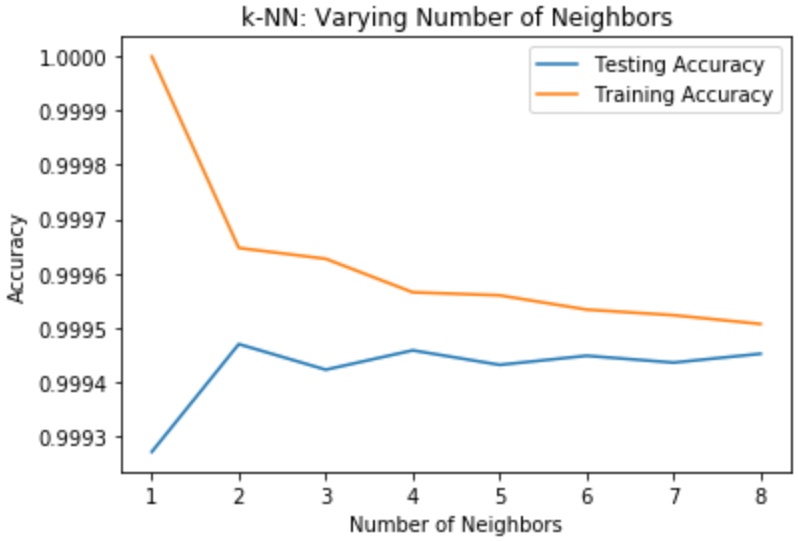
KNN - KNeighborsClassifier()

The KNeighbors Classifier is a supervised learning technique that learns from the labelled inputs to form classification clusters. The k represents the number of points, or "nearest neighbors" to consider when forming the clusters.

- Parameter tuning
    - The default value is k or n_neighbors=5. We use the accuracy score to evaluate the train and test predictions with varying values for k.
    - The run time is over 5 hours, but using PCA reduces it to about 3 minutes, so it will be used to test parameter selections using GridsearchCV.

https://github.com/cruzer42/Capstone-Project-1-for-Springboard/blob/master/Capstone%20Project%201%20knn-findk_parameter.ipynb

The default value 5 looks like a reasonable choice, as the slope of the train vs test accuracy slope is at a juncture where training accuracy decreases while test accuracy levels off. Perhaps selecting 2 preserves the highest train accuracy given the test level, so this will be compared.



The test result did not improve, but using PCA made the run time reasonable to obtain a gridsearch selection between 2 and 5. However, the best estimator was selected as 5 which is the default.

https://github.com/cruzer42/Capstone-Project-1-for-Springboard/blob/master/Capstone%20Project%201-loadfeatures-knn-pca-gridsearch-k2-5.ipynb

SVM - svm.SVC() - C-Support Vector Classification

Support Vector Machine (SVM) is a linear model for classification and regression problems. At a high level, it creates a line or hyperplane to classify data. The initial classifier of this model we consider, Support Vectors Classifier (SVC) tries to find the best hyperplane to separate the different classes by maximizing the distance between sample points and the hyperplane.

- Parameter tuning
    - The run time with default parameters is over 4 hours, and over 5 hours when combined with PCA. A result could not be obtained for SMOTE, likely due to non-convergence.
    - Due to the long run time, it did not seem plausible to run GridSearchCV on parameters to test defaults such as the regularization c=1.0. Interestingly, the alternatives to the default model defined as kernel='rbf' includes a value 'linear' which appears to be equivalent to using the classification model LSVC instead.
    - The max_iter=-1 default means there is no max so it runs until convergence is found. This parameter was tested with max_iter=1000 and the rbf kernel, but results did not converge. Results were obtained using LSVC instead.

LSVM - svm.LinearSVC() - Linear Support Vector Classification

Similar to SVC with parameter kernel='linear', but implemented inrun terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples. Test results show that we do not get the same results as simply using the kernel and max_iter settings with SVC as running LSVC.

- Parameter tuning
    - The run time with default parameters is about 14 minutes, compared with over 4 hours using SVM.
    - GridsearchCV results will be reviewed for a selection of parameters to determine whether incremental improvements are worth pursuing.

Inferential

GNB - GaussianNB() - Naive Bayes

Naive Bayes classifier calculates the probability of an event by calculating prior probability for given class labels, conditional probability with each attribute per class, and multiplying out class and prior.

The Gaussian Naive Bayes Classifier was included in order to represent an additional class of models. Although not necessarily suited for a binary classification problem, it is better than the alternative Multinomial Naive Bayes, which by its name would not be applicable in the binomial case. There does not seem to be much opportunity for parameter tuning, and the results did not score very well on the metrics. The run time is fast but that is not enough to justify spending too much effort to use this model given the poor results.

Unsupervised

KMEANS - KMeans()

KMeans is an unsupervised clustering algorithm which divides observations into k clusters. The data has no labels, as would be required for supervised learning. This model was selected for comparison and contrast with its supervised counterpart, KNN.

- Parameter tuning
    - The default value is k=5. For binary classification, we must use k=2 to match the only possible outcomes.
    - The run time is only a few minutes, but Precision and Recall scores are approaching zero so the model does not appear useful at first.
- Notable results
    - When adding PCA to the pipeline, the True vs False Negative totals flip, but the False Positives are extremely high. With SMOTE, the False Positives are nearly

resolved and False Negatives are not quite as good as most other models, but within reason. The short run time makes it a candidate for GridsearchCV tuning.

Neural Network

MLP - MLPClassifier() - Multi Layer Perceptron

The Multi Layer Perceptron is a supervised learning algorithm that learns a function by training on a dataset. The MLP consists of three or more layers, an input and output layer, with one or more hidden layers.

- Notable results
    - The initial metrics using default parameters are not far off from the top model RFC, with run time of about 15 minutes compared to about 7 for RFC.
    - Including PCA yields similar results but in about half the time. SMOTE renders the False Positives not worth the tradeoff of improving False Negatives, if balance is important, and it increases the run time to over 3 hours.
    - There is potential to improve the model with tuning and learning, which can be more complex than some of the other models. This will be a topic for further completion.

Ensemble Bagging

Ensembled algorithms are those which combine more than one algorithm of same or different kind for classifying objects.

RFC - RandomForestClassifier()

Random Forest Classifier creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object.

This model had the best results in terms of metrics that measure False Positive and False Negative rates. It will be used to illustrate model tuning from normalization and feature selection, to parameter tuning and measuring performance.

- Parameter tuning
    - The default value model is indicated by criterion='gini'. Test results indicate that 'entropy' produces slightly better results.
    - The run time is only about 7 minutes, making it a reasonable candidate for exploring the model parameter optimization.
    - PCA and SMOTE diminished the original result, so neither will be selected for the final model.
    - Parameter tuning with respect to class weight balancing led to discovery of a Balanced RFC, which was included in the testing for comparison.
    - The default n_estimators=10 was changed to n_estimators=100 in a higher version. The new default will be used in the final model.
    - The max_depth=None default will be change to max_depth=30 based on GridsearchCV results of parameter tuning.

BRFC - BalancedRandomForestClassifier()

A balanced random forest randomly under-samples each boostrap sample to balance it. This sounds like an intuitive fit for a binary classification problem, but the results did not pan out for this or any other methods designed to address imbalanced data. The performance metric results were poor but run time was fast at around 4 minutes. However, using SMOTE increased run time to over an hour, and even though PCA alone has little run time effect, when combined with SMOTE as well run time reached nearly 6 hours.

XGB - GradientBoostingClassifier()

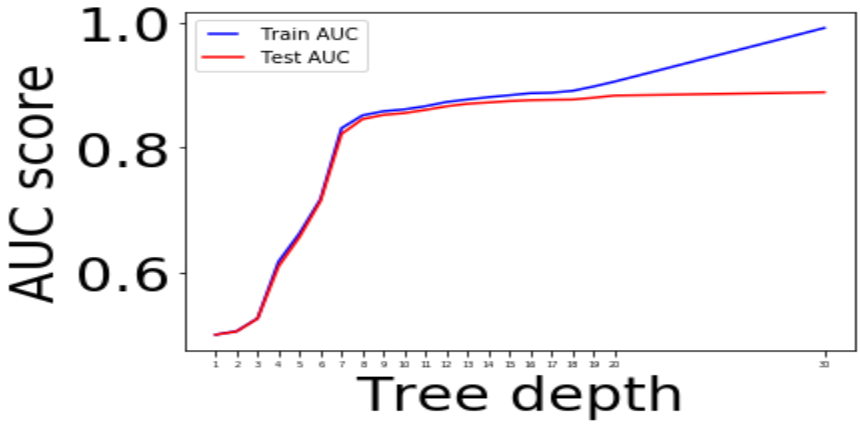
XGB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage n_classes regression trees are fit on the negative gradient of the binomial or multinomial deviance loss function. Binary classification is a special case where only a single regression tree is induced. The performance metrics were generally poor and run time starting at about 20 minutes, and increasing to 30 and 40 minutes with PCA and SMOTE respectively. This model does not appear to have much promise with respect to dramatic improvements from parameter tuning.

Final Model Tuning

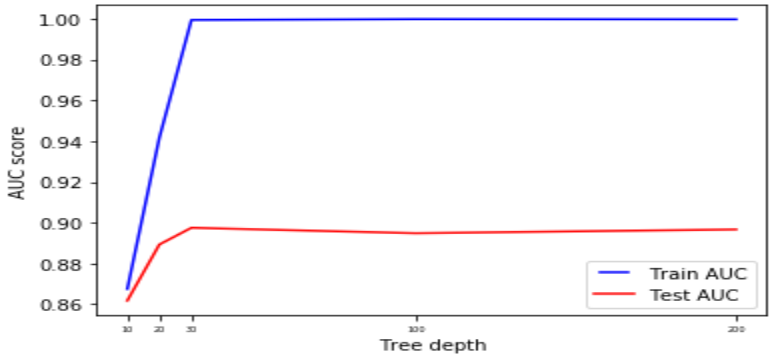
The Random Forest Classifier was tuned over several iterations of GridSearchCV, taking advantage of best parameter selections from prior runs. In addition to striving for few False Negatives and False Positives, we need to perform a test to measure whether the model is overfitting or underfitting.

- Parameter tuning
    - The default max_depth=None is reviewed as we measure the train and test metric AUC over a range of max_depth values. The value represents maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
    - Note in some runs, the max_depth best_estimator result selected as 30, and other times 20. This indicates that the difference in measured result may be very small, so we can review other tradeoffs.
- Test for Overfitting/Underfitting
    - By graphing the train and test result for the AUC metric over a range of max_depth values, we can visualize the part where overfitting is indicated by train results outpacing test results at the margin.

https://github.com/cruzer42/Capstone-Project-1-for-Springboard/blob/master/Capstone%20Project%201%20Testing-rfc_gridsearch.ipynb



- Note the same graphs with a different scale, showing that 30 is an inflection point where both sets of results level off



- Both views depict the train results moving away from the test result, so 20 is the point that trades off potential overfitting for a small gain in metrics.

Final Result

The final result is evaluated in terms of gains in metrics, attributed to features and tuning. The model was selected for tuning among 10 in a baseline test.

The final **RandomForestClassifier()** model uses non-default values criterion='entropy', as well as an update to n_estimators=100 to match the new default, along with max_depth=20.

See the final result appended to the feature selection result table from above. We have improved the results using feature selection and model tuning. Cumulative Recall score improved .05, with F1 and ROC-AUC at .04, and .03 improvement. False Negatives decreased from 546 to 421, and False Positives from 51 to 26, for the test of sample size in excess of 1.9M.

Confusion Matrix	[[1906300 51]	[[1906288 63]	[[1906322 29]	[[1906325 26]	-25 FP diff
	[546 1889]]	[487 1948]]	[445 1990]]	[421 2014]]	-125 FN diff
Model	start features	plus Change balance	plus Time conversion	rfc best_params	Diff best - start
Accuracy score	0.9997	0.9997	0.9998	0.9998	0.0001
F1 score	0.8635	0.8763	0.8936	0.9001	0.0366
Precision score	0.9737	0.9687	0.9856	0.9873	0.0135
Recall score	0.7758	0.8000	0.8172	0.8271	0.0513
ROC-AUC score	0.8879	0.9000	0.9086	0.9135	0.0257

https://github.com/cruzer42/Capstone-Project-1-for-Springboard/blob/master/Capstone%20Project%201-loadfeatures-rfc-gridsearch.ipynb

Future Work

Parameter Tuning

Other models

Parameter tuning was applied to LR and KNN models with minor gains in metrics. It made sense to review key parameters for overfit/underfit by comparing train vs test results over a range of values. The same could be applied to the remainder of the models, some with a great deal of complexity. In particular, tuning a Neural Network such as MLP and variations of XG Boost have worked out for some Kagglers.

Sampling data

Train, Test, Split test size was held constant, and default parameters were used for SMOTE and PCA. Parameter tuning could be applied to each to ensure optimal selection.

Ensembles possible with combination of models

Alternate criteria and ensemble combinations could be interesting to pursue. For example, it might be plausible to focus on minimizing False Negatives by prioritizing Recall at the expense of Precision, or combine models.