Josh Thomason

Capstone: Credit Card Fraud

Final Report:

Problem:

Credit-card fraud is a big problem for the customers of banks, not only that but lowers the customers' trust in their bank. If gone unchecked could cause very serious problems for people being taken advantage of. So, banks need to create a way to try and predict what transactions could be/are fraudulent before the problem becomes too big. That is where we come in. To try and solve this problem I used a data set compiled of approximately 280,000 credit card transactions, that are classified as non-fraudulent (0) or fraudulent (1). It is made up of 2013 credit-card transactions in Europe.

Process:

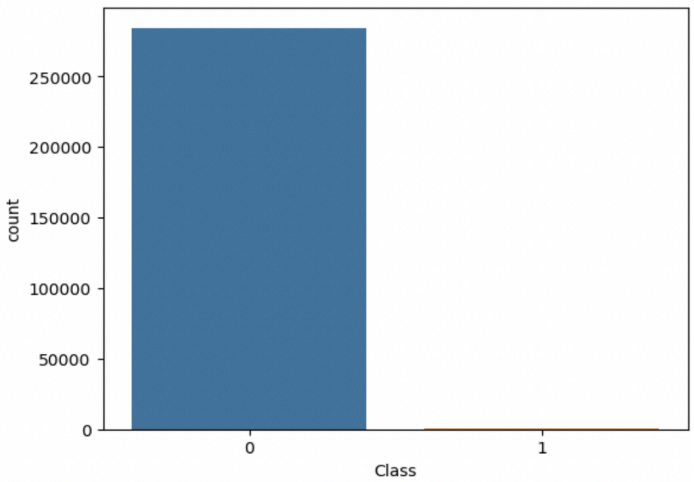
Data Wrangling:

This step of the project was pretty simple for this data set. There are many different columns, but the most important ones are; ‘Time’, ‘Amount’, and ‘Class’. ‘Class’ is broken down to two values Class 0 (non-fraudulent), and Class 1 (fraudulent). There are also columns of ‘V1’ - ‘V28’ that were acquired beforehand using a PCA. The shape of this data set is (284807, 31). Each of the columns has a non-null count of 284807, meaning that none of the columns or rows have any null values, no need to replace or drop data. I Wanted to see the distribution of the ‘Class’ values in the data. Split it into two data sets: a non_fraud and fraud_trans data set. Found that 99.83% of the data falls into Class 0, while only With an understanding of our columns and not much data to clean up we can move to the visualization step.

EDA (Exploratory Data Analysis):

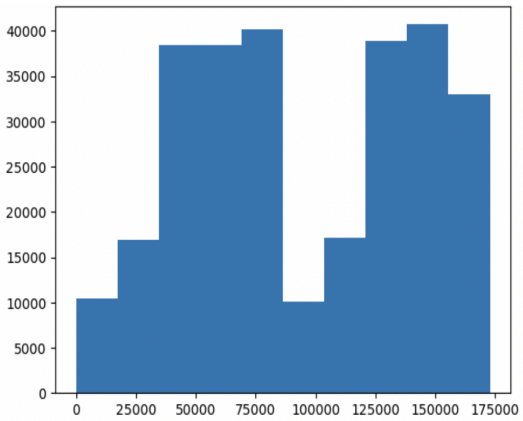
This step of the capstone was not as helpful as one would wish. This is because of our heavily imbalanced classes, and the fact that there were only 2 possible classes.

Histogram of ‘Class’ Distribution



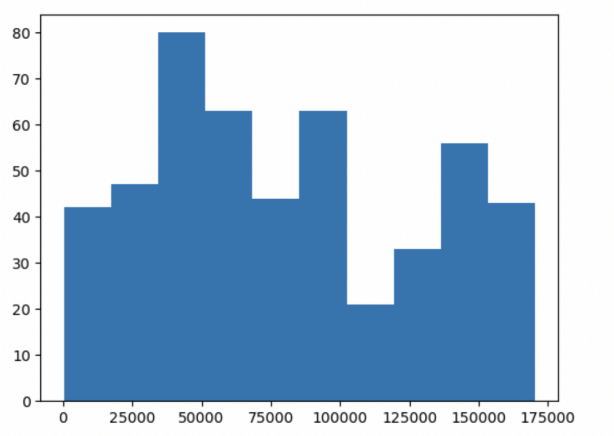
Also, our ‘Time’ and ‘Amount’ columns had very high ranges and large standard deviations. Despite that, splitting the data set into non_fraud and fraud_trans allowed me to see a very minor relationship between ‘Time’, ‘Amount, and ‘Class’.

non_fraud[‘Time’] Histogram:



Description: Mean = 94838.20
Standard Deviation = 47484.02
Min = 0.00
Max = 172792.00

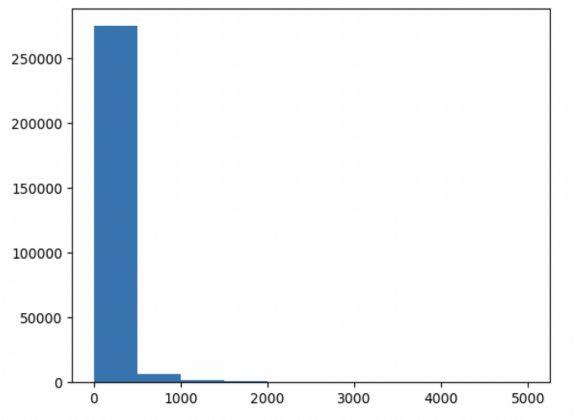
fraud_trans['Time'] Histogram:



Description: Mean = 80746.81
Standard Deviation = 47835.37
Min = 406.00
Max = 170348.00

Through the use of histograms you can see some very little differences in the time value when split into fraudulent and non-fraudulent classes. But, if you look at the description of the variables grouped by class, you can see some minor relationships. Like I said it's minor, but a fraudulent transaction on average will be 14,000 seconds before a non-fraudulent transaction. This could be due to a much larger sample size for the non-fraudulent class. But, we can't count out that there is a relationship, so, the 'Time' column will be used in the model building.

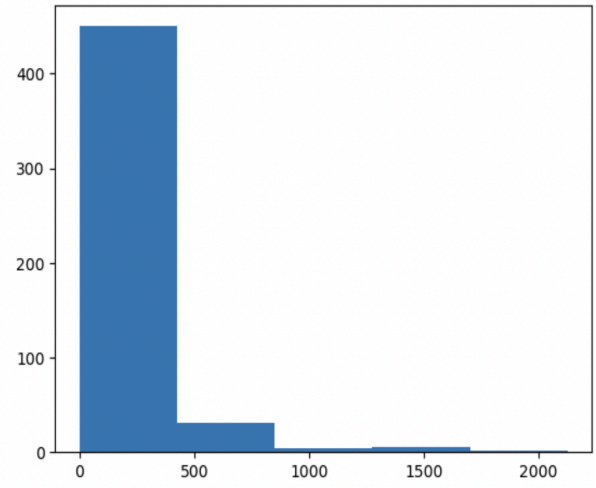
non_fraud['Amount'] Histogram:



Description: Mean = 88.291

Standard Deviation = 250.105
Min = 0.00
Max = 25691.160

fraud_trans['Amount'] histogram:



Description: Mean = 122.211
Standard Deviation = 256.683
Min = 0.00
Max = 2125.870

Something similar happened with the 'Amount' columns when you group them by their 'Class'. Again, there is a large difference in the sample size between these two histogram/df's. Although, we cannot count out a relationship. Through the description of the statistics and the histograms, we can see that fraudulent transactions will on average be $34 more than a non-fraudulent transaction. Therefore, 'Amount' will be another valuable variable for our model building.

We also ran a heatmap to try and visualize any correlations between the variables, mostly looking for correlation between variables and the 'Class' (our dependent variable) columns. There were no meaningful connections between the data points, so, wasn't the best visualization metric for us.

Preprocessing:

Through our exploratory data analysis we found that some things need to happen to our data and variables. Although, with the way our data set was set up there was not much that needed to be done in this step. One of the common steps to take is to create a dummy variable for our categorical data. But, our categorical data 'Class' was already set up to take on 1 of 2 values 0 (non fraudulent), and 1 (fraudulent). So, there was no need to create any dummy variables for our data set. We then split our data into training and testing sets, with the test_size set = 0.25, and our random_state = 14. The data did have a lot of numerical values, so we passed it through a StandardScaler(). This standardized our data to make it better suited for modeling.

```
X = credit_card.drop(['Class'], axis =1)
y = credit_card['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 14)
Then standardized it using StandardScaler() and fit_transform it to our X_train data.
Also, transformed our X_test data.
```

Model-Building:

Model 1 (KNN):

The first model I built was a KNN (KNeighborsClassifier()) model. I used a random search cv to find the best parameters for our model. The parameter we were testing for was 'n_neighbors'. I initiated the search through a range of n_neighbors of 1-10, cv = 5, and set our scoring method to 'roc_auc' because that is the best metric for imbalanced data. Then fit it to our X and y data. This returned a best parameter of n_neighbors = 1.

I then initiated our KNN model setting the parameters to n_neighbors = 1, and fit it to X_train & y_train. Used the .predict() method to test the metrics of the model. Then I went through and checked those metrics. This returned a roc_auc_score = 0.912.

This is not a bad roc_auc_score, but due to the high imbalance of our data we want a higher score than that.

Model 2 (Random Forest):

The second model I built was a Random Forest Classifier. Again, used a random search cv to find the best parameters for this model. I changed the cv = to a repeated stratified kfold. Then fit the random search with the X & y data. This returned our best parameters set to n_estimators = 256, min_samples_split = 4, min_samples_leaf = 2, max_depth = 7.

I then initiated the Random Forest model with those optimal parameters. Then, I fit it to our X_train and y_train data. Used the .predict() method to find the metrics associated with the model. I checked the roc_auc_score for this model and it came out to be = .895.

Again, not a terrible roc_auc_score, but not quite what we were looking for in a final model.

Model 3 (Logistic Regression):

The third model I tested/built was a Logistic Regression model. This time I used a grid search cv to try and find the optimal parameters. The parameter I tested was the C value, the param_grid = {'C': [0.01, 0.1, 1, 10, 100, 1000]}. For the cross-validation (cv), I used the KFold method for this search, and scoring again set to 'roc_auc'. I fit it to the X & y data, this returned the best parameters of C = 0.01.

I built a Logistic Regression model with the parameter of C = 0.01. Then, fit it to the X_train and y_train data. Used the .predict() method to see the metrics of the model. Returned and roc_auc_score = 0.80.

This roc_auc_score is nothing close to the value we are trying to get. Obviously, there is a problem that I need to try and solve.

Problem with the first 3 models:

The problem we are facing is trying to build a model that is able to predict if a transaction is either fraudulent or non-fraudulent. This is a problem because our data is highly skewed, with more than 99.5% of all of our data points being classified as non-fraudulent. There are two methods I used to try and solve this.

Through quite a bit of research I found that resampling methods are a good way to work around highly imbalanced data, like what we have. The two types of resampling methods I looked at were undersampling and oversampling.

Undersampling:

This method helps us resample and fit our new X_train and y_train data sets. This method makes it so that there are equal amounts of data points in each of our classes. But, since 'Class' 0 was much higher than 'Class' 1, it brings down the number of 'Class' 0 points to match that of 'Class' 1. Now, the data points are equal between the classes.

Class 0: 377 (data points)
Class 1: 377 (data points)

It is also computationally inexpensive because there are less data points to iterate and test over.

Building Models on our Undersampled train sets (Using RandomUnderSampler() method):

Undersampled Model 1 (LogisticRegression):

The first model I built using the new undersampled data was a Logistic Regression model. Again, I used a grid search cv to find the optimal parameters. Used the repeated stratified kfold for our cross-validation (cv) method. I used this search to test the 'C' parameters,

the same as before, {'C' : [0.001, 0.01, 0.1, 1, 10, 100, 1000]}. Fit the search to the undersampled X_train and y_train data (X_train_unsam, y_train_unsam respectively). Again, returned our best parameter 'C' = 0.01, but it was trained using the resampled data.

I then initiated the model with C = 0.01, and fit it to X_train_unsam & y_train_unsam. We are still trying to find the optimal roc_auc_score, so we used the .predict & .predict_proba to best find that metric. For this undersampled model we tested for train_roc and test_roc. These returned values of…..

Train_roc = 0.986
Test_roc = 0.975

These scores are much higher than the previous ones we built thanks to the undersampling method, and can be considered a good predictive model.

Undersampled Model 2 (KNN):

The second model I built/tested with the resampled data was the KNN (KNeighborsClassifier). This time I used a grid search cv to test for the best performing parameters. Again, tested for 'n_neighbors' (in a range of 1-50), KFold method for cv, scoring = 'roc_auc'. This returned our optimal parameter of 'n_neighbors' = 33.

Then initiated our KNN model with the parameter 'n_neighbors' = 33, and fit it to the resampled train sets (X_train_unsam, y_train_unsam). Similar to the Logistic Regression model I used .predict() and .predict_proba() to best find the roc_auc_score for our resampled train and test sets. This returned values of….

Train_roc = 0.984
Test_roc = 0.958

Again, these scores are much more acceptable compared to the previous KNN model, but not quite as high as the Logistic Regression model we previously built.

Undersampled Model 3 (RandomForest):

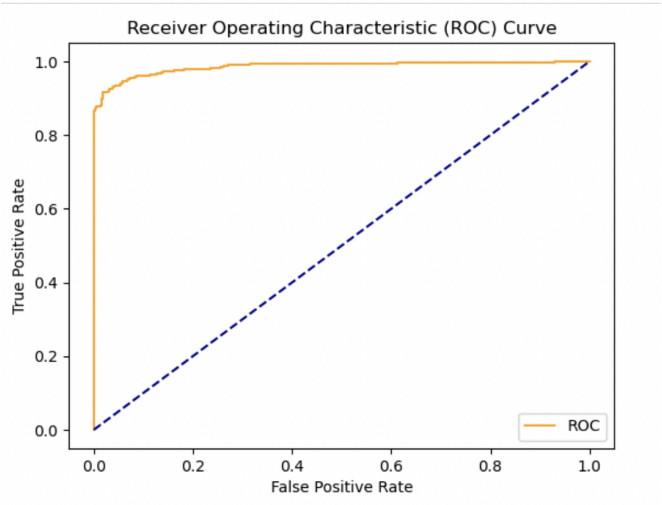
The third model I built/tested with the resampled data was a Random Forest Classifier. Used a grid search cv to test for the best parameters to use in building the model. Since a Random Forest is computationally more expensive than the other models I used a basic cv, setting it equal to 5, and again scoring = 'roc_auc'. This returned the best parameters of… n_estimators = 65, min_samples_split = 40, min_samples_leaf = 20, max_features = 20, max_depth = 50, bootstrap = True.

Then, built a Random Forest model with those parameters and fit it to X_train_unsam & y_train_unsam. Used the same predictive methods to find the roc_auc_score metric for our train and test sets. This returned values of ……

Train_roc = 0.987
Test_roc = 0.969

Much higher than previous scores for our first Random Forest. Although, it is more computationally expensive compared to other models (mostly Logistic Regression), and the test_roc is lower than that of the Logistic Regression model. Therefore, it is a good model, but the logistic regression model is still better for the time, and predicting on the test set.

Roc Curve for the best model (Logistic Regression):



Optimal threshold = 0.3503

Resampling Method 2 (OverSampling):

Oversampling is similar to undersampling in the sense that you fit and resample the X_train and y_train data sets to get an equal number of points for each class value. The way it is different is because instead of coming down to the number of points in 'Class' 1, it brings Class 1 up to the amount of points in Class 0.

Class 0: 213228
Class 1: 213228

The problem with this method of resampling is that it is much more computationally expensive compared to the undersampling method. Therefore, it makes sense to only use this on simple models (like Logistic Regression). Although, it doesn't leave anything out like undersampling does.

Building 1 model (Logistic) based on our oversampled data (Using SMOTE()):

Oversampled Model, Logistic Regression:

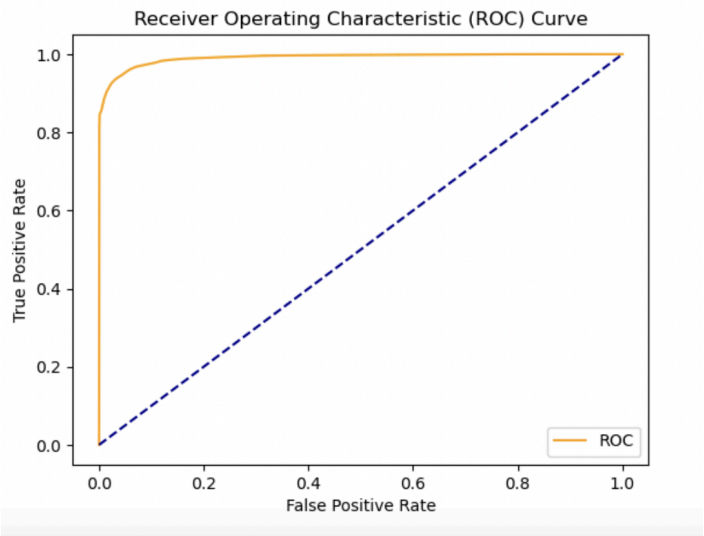
Built a grid search cv to find the best parameters, {'C' : [0.01, 0.1, 1, 100, 1000]}. Again, used KFold as our cv, and scoring = 'roc_auc'. Fit it to our resampled data (X_train_smote, y_train_smote). This returned our best parameter of C = 1000.

Built a model with that parameter of C = 1000, and fit it to X_train_smote & y_train_smote. Used the same predictive methods to try and find the metrics we wanted for the model (roc_auc_score) for the train and test sets. Returned scores of…..

Train_roc = 0.911
Test_roc = 0.978

These are really good scores for our model, slightly higher than the undersampling method. The only issue is that this method is more computationally expensive, and would require more time to create and train this model.

Roc Curve of this Oversampled Logistic Regression Model:



Optimal Threshold = 0.4322

Findings:

We found that over the three models we tested/built (KNN, RandomForest, LogisticRegression) the Logistic Regression is the simplest and performs the best. We also found that the resampling techniques are crucial to building a good predictive model given the high class imbalance. The SMOTE method is the best at finding the best model, given that time and resources are a given. And, that the undersampling method is good for finding a simple easy to test/train model when there is a time/resource constraint, while still having good predictive capabilities for the class classification.

Recommendations:

Bigger Company:

For a company with more resources I would recommend the SMOTE (oversampling method) version of our Logistic Regression model. Given the higher resources they can test and train the model easier than their smaller counterparts. It has the highest predictive capabilities of

any of our models that we built. If it is a bigger company, with a higher number of customers and transactions happening, I recommend they send out an email, text or call the concerned party and have them verify the transaction, so they can save a little time on that aspect.

Smaller Company:

I would recommend the undersampled (RandomUnderSampler) Logistic Regression Model. It has high predictive capability given by the high roc_auc_score. It is computationally inexpensive, better for companies with lower resources and workers. If the company is smaller with fewer customers, I would recommend that someone take a closer look at the transactions flagged as fraudulent and go from there.

.