Machine Learning Engineer Nanodegree

Capstone Project: Credit Card Fraud Detection Model

Akbarali Shaikh July 16, 2020

Table of Content

1. Project Overview

Banking industry works on credit and income from interest which can be in a form of mortgage, other types of loan or from Credit Card. As cost of having credit card reduced it got broadly accepted in society and many people started having credit card. As the credit card reached the masses, new ways of fraud also came to existence. Today, CC fraud is a very big problem for a banking industry and it is plaguing financial industries for years, some countries it is more less problem than others

Solving this problem could bring about reduced volume of fraudulent transactions in financial industries hereby saving them a huge volume of money lost and this problem can be avoided by using predictive analytics where machine learning algorithms are used to detect fraud patterns and determine future probabilities and trends.

The dataset to be used for this project was gotten from data.world

2. Problem Statement

Jus by looking at the transaction, it is often difficult know which is fraudulent or genuine. We have a binary classification problem i.e. either a transaction is fraudulent or genuine. Objective is to create a model that accurately predicts whether a transaction is fraudulent or not.

To solve this problem, first we will baseline the model and than run a predictive model on different machine learning algorithms on

same dataset to see which will give the best performance which is better than the others,

These algorithms include Naïve Bayes Classifier, SVC, GaussianNB, LogisticRegression, KNeighborsClassifier, DecisionTreeClassifier, RandomForestClassifier, GradientBoostingClassifier, BaggingClassifier, ExtraTreesClassifier, XGBClassifier and AdaBoostClassifier.

Dataset is normalized and all columns names changed before it is shared, it could also be a case where some of the features were reduced from original dataset, no other form of reduction would be made going forward. All features provided would be used in building the model.

3. Metrics

In a binary classification problem such as this, a model classifies examples as either positive (fraudulent) or negative (genuine). The decision made by the model, either positive or negative can be represented in a structure known as confusion matrix. This confusion matrix has four elements that define it, contextually they are:

- True Positive (TP) – An example where a transaction is fraudulent and is classified correctly as fraudulent.

- False Positive (FP) – An example where a transaction is genuine and is classified as incorrectly as fraudulent.

- True Negative (TN) – An example where a transaction is fraudulent but is classified incorrectly as genuine.

- False Negative (FN) – An example that is genuine and is classified correctly as fraudulent.

In most binary classification problems, accuracy is used as a metric for evaluating models and is given as follows:

$$\text{Accuracy} = \frac{True\ Positive + True\ Negative}{Total\ Observations}$$

Accuracy is a measure of how well a model accurately predicts true examples as positive and false examples as negative.
Now in situations like this where there is a class imbalance ratio (negative examples outnumber positive examples by a very wide margin), the performance of a model is not reflected by accuracy because there are not enough training points for the positive class (fraudulent transactions).

Given the class imbalance ratio in the dataset, there are other evaluation metrics that take such class imbalance into consideration:

i. Precision
ii. Recall

Recall measures the fraction of fraudulent transactions that are correctly predicted while measures that fraction of cases predicted to be fraudulent that are truly fraudulent.

Precision and Recall are defined as follows:

$$\text{Precision} = \frac{TP}{TP+FP} \qquad \text{Recall} = \frac{TP}{TP+FN}$$

Since Precision & Recall don't account for true negatives (cases where a transaction is fraudulent but classified as genuine by the model) the choice for using these two elements) seems reasonable given that there are lot more cases of genuine transactions than fraudulent.

We also need to aware of Type I and Type II error.

False Positive Rate - Type I error When we predict something when it isn't we are contributing to the false positive rate. We can think of it as a fraction of false alerts that will be raised based on your model predictions. for e.g. predicting fraud transaction as genuine

False Negative Rate - Type II error When we don't predict something when it is, we are contributing to the false negative rate.

We can think of it as a fraction of missed fraudulent transactions that your model lets through. for e.g. predicting genuine transaction as fraud

Recall It is a go-to metric, when one really care about catching all fraudulent transactions even at a cost of false alerts. Potentially it is cheap for you to process those alerts and very expensive when the transaction goes unseen.

Precision It measures how many observations predicted as positive are in fact positive. When raising false alerts is costly, when one want all the positive predictions to be worth looking at you should optimize for precision.

Accuracy It measures how many observations, both positive and negative, were correctly classified. One shouldn't use accuracy on imbalanced problems. Then, it is easy to get a high accuracy score by simply classifying all observations as the majority class. For example in our case, by classifying all transactions as non-fraudulent we can get an accuracy of over 0.9

F beta score It combines precision and recall into one metric. The higher the score the better model. The beta value greater than 1 means we want our model to pay more attention to the model Recall as compared to Precision. On the other, a value of less than 1 puts more emphasis on Precision.

F1 score (beta=1) It's the harmonic mean between precision and recall

F2 score (beta=2) It's a metric that combines precision and recall, putting 2x emphasis on recall - consider using it when recalling positive observations (fraudulent transactions) is more important than being precise about it

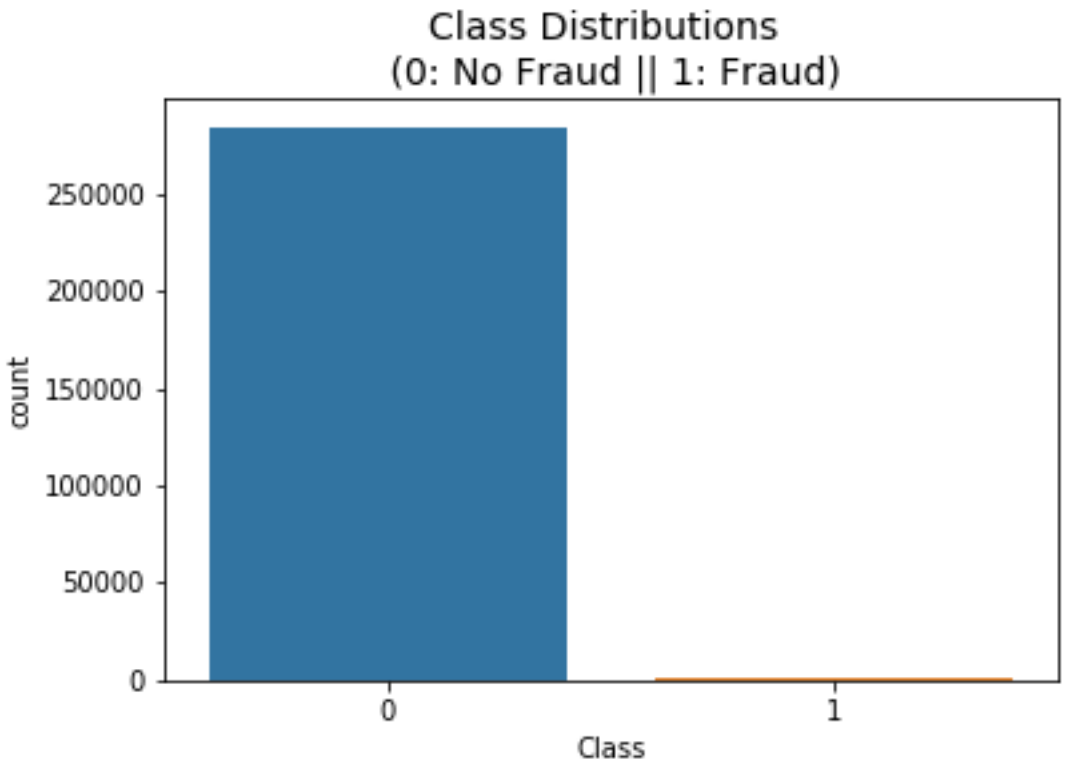
4. Analysis

Data Exploration

The dataset used is anonymized set of credit card transactions labelled as fraudulent or genuine. Due to confidentiality issues, the original features and more background information about the data were not provided.
The dataset presents transactions that occurred in two days, where there are 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for .17% of all transactions.

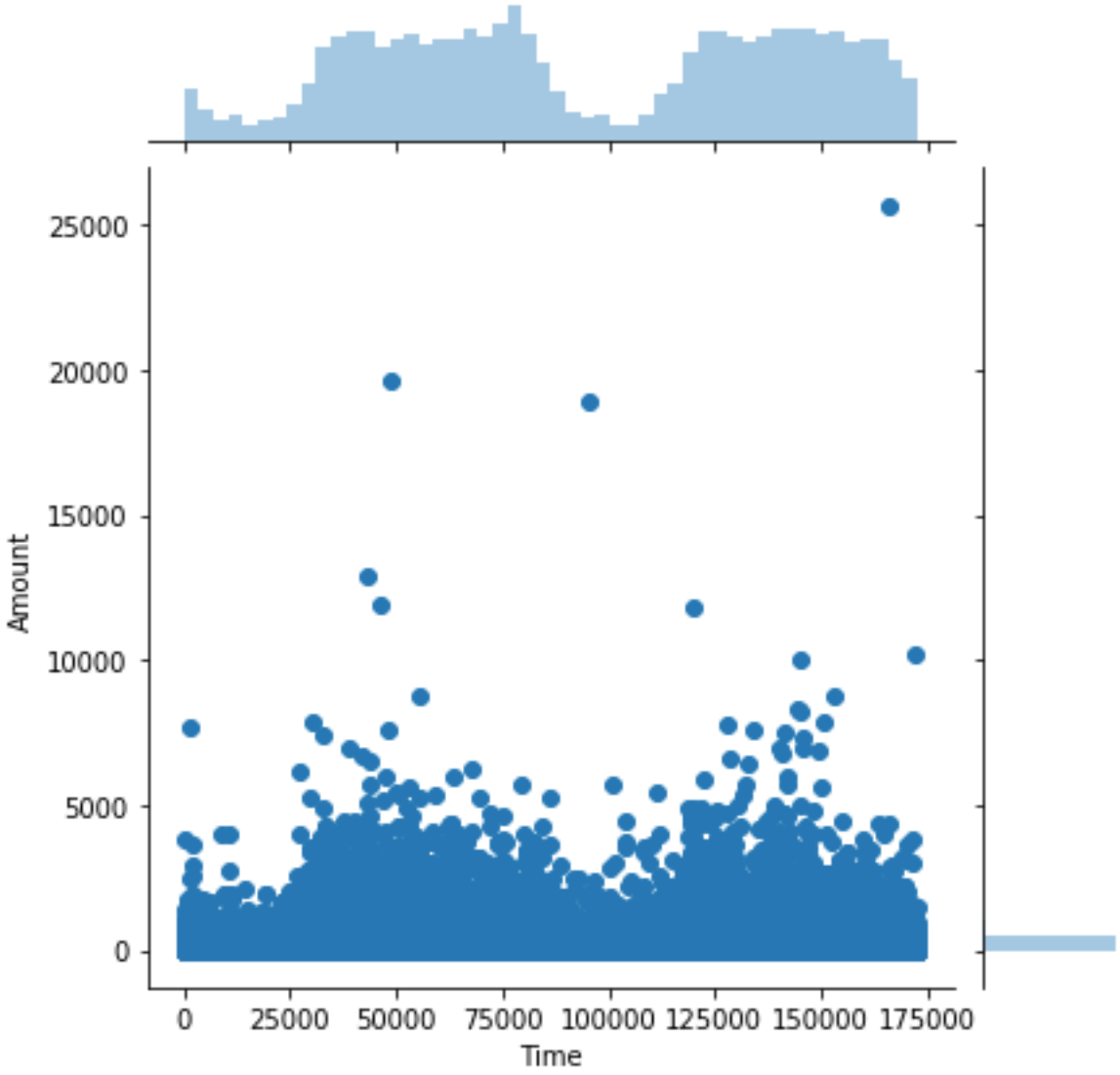
The dataset contains 31 numerical features. The first 28 features are labelled V1, V2....V28 and these are principal components obtained with Principal Components Analysis of the raw/original data.

Exploratory Visualization
Fig 1. A plot showing the ratio of fraudulent transactions to genuine transactions, from this visualization it is obvious that fraudulent cases are more and if analysed directly model will be biased.

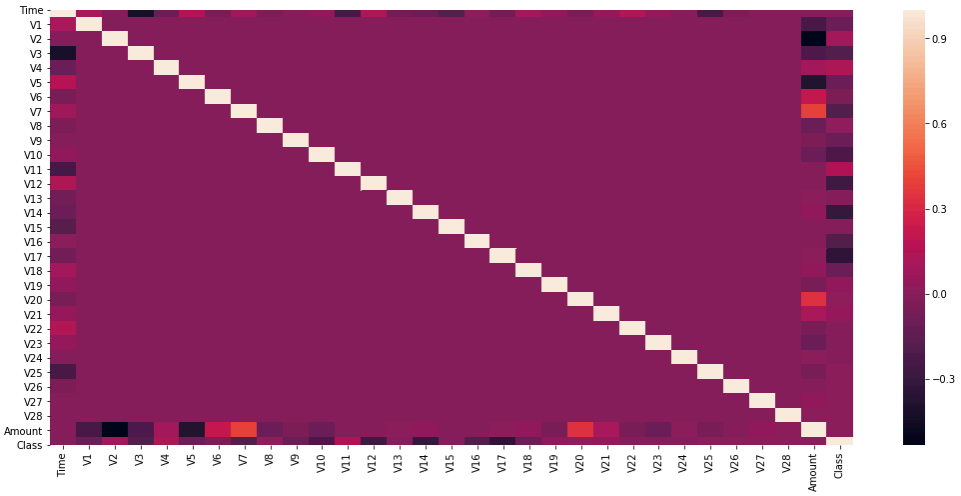



```
# The classes are heavily skewed we need to solve this issue later.
print('No Frauds', round(df['Class'].value_counts()[0]/df.shape[0] * 100,2), '% of the dataset')
print('Frauds', round(df['Class'].value_counts()[1]/df.shape[0] * 100,2), '% of the dataset')
```

```
No Frauds 99.83 % of the dataset
Frauds 0.17 % of the dataset
```



Correlation

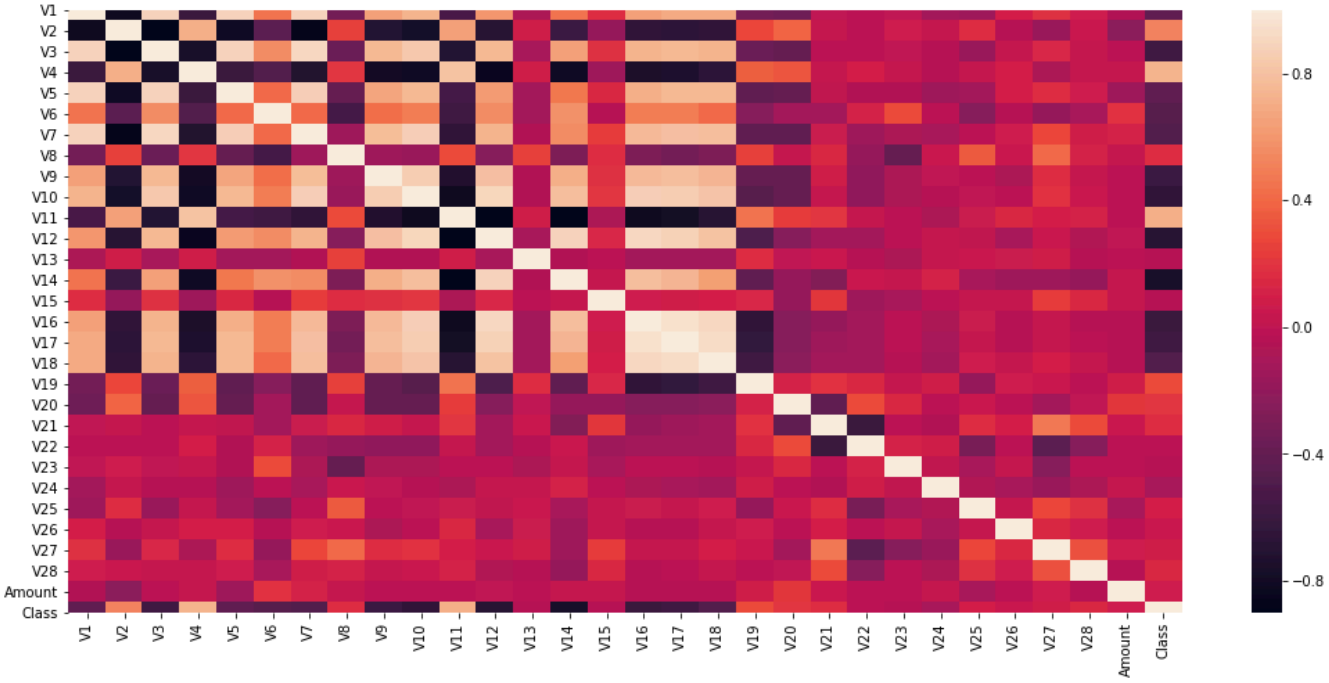


5. Modelling

Before we proceed, lets try baselining the model.
Step 1: Run a Logistic Regression model on as is data - baseline the model and scores.
Step 2: Now we implement SMOTE and Minmax scaller,

- SMOTE will increase the minority class, this will have equal number of fraud and genuine transaction

Step 3: Visualise and check if we have any correlation



Step 4: Run minmax scaller will support to have values columns values between -1 and 1

Step 5: Run a Naive predictor, in this algorithm we assume all values is zeros than what will be the accuracy and f1 score of the model

```
Naive predictor accuracy: 0.998
Naive predictor f1-score: 0.000
```

Steps 6: Run the model with no hyper parameter tuning to check the accuracy and f1 score to check the against the baseline

	Model Name	Accuracy	F1 Score	Recall	Precision	F1 beta
11	XGBClassifier	0.999587	0.845238	0.755319	0.959459	0.808231
5	RandomForestClassifier	0.999571	0.838323	0.744681	0.958904	0.799649
9	BaggingClassifier	0.999571	0.838323	0.744681	0.958904	0.799649
10	ExtraTreesClassifier	0.999539	0.826347	0.734043	0.945205	0.788225
3	KNeighborsClassifier	0.999508	0.812121	0.712766	0.943662	0.770796
8	SVC	0.999460	0.808989	0.765957	0.857143	0.791878
4	DecisionTreeClassifier	0.999301	0.755556	0.723404	0.790698	0.742857
12	AdaBoostClassifier	0.999254	0.728324	0.670213	0.797468	0.704819
0	Logistic Regression - Baselining - Train	0.999174	0.666667	0.553191	0.838710	0.617916
1	Logistic Regression - Baselining - Test	0.999159	0.697318	0.579618	0.875000	0.646802
6	GradientBoostingClassifier	0.999127	0.640523	0.521277	0.830508	0.588725
2	LogisticRegression	0.999063	0.581560	0.436170	0.872340	0.515474
7	GaussianNB	0.978228	0.099803	0.808511	0.053184	0.150564

We can see from above image - 8 models have worked better than the baseline. Difference in accuracy between top 4 models is 0,000048 - not much too compare but we need to keep in mind that the dataset is biased hence we should also take F1 score in consideration. Difference between the top 4 F1 score is 0,018891. Since all top 4 models accuracy and F1 range is in very narrow range hence, implemented hyper-parameter tuning on all the 4 models and tested the model with test data.

Model was trained on trained data and tested in validation set. We will use the test data on hyper-parametered models on all the top 4 models

	Model Name	Accuracy	F1 Score	Recall	Precision	F1 beta
0	XGBClassifier	0.999587	0.845238	0.755319	0.959459	0.808231
1	RandomForestClassifier	0.999571	0.838323	0.744681	0.958904	0.799649
2	BaggingClassifier	0.999571	0.838323	0.744681	0.958904	0.799649
13	ExtraTreesClassifier - Test Set	0.999564	0.858131	0.789809	0.939394	0.830500
14	XGBClassifier - Test Set	0.999564	0.858131	0.789809	0.939394	0.830500
15	Random Forest - Test Set	0.999564	0.858131	0.789809	0.939394	0.830500
3	ExtraTreesClassifier	0.999539	0.826347	0.734043	0.945205	0.788225
4	KNeighborsClassifier	0.999508	0.812121	0.712766	0.943662	0.770796
5	SVC	0.999460	0.808989	0.765957	0.857143	0.791878
16	BaggingClassifier - Test Set	0.999436	0.810036	0.719745	0.926230	0.772751
6	DecisionTreeClassifier	0.999301	0.755556	0.723404	0.790698	0.742857
7	AdaBoostClassifier	0.999254	0.728324	0.670213	0.797468	0.704819
8	Logistic Regression - Baselining - Train	0.999174	0.666667	0.553191	0.838710	0.617916
9	Logistic Regression - Baselining - Test	0.999159	0.697318	0.579618	0.875000	0.646802
10	GradientBoostingClassifier	0.999127	0.640523	0.521277	0.830508	0.588725
11	LogisticRegression	0.999063	0.581560	0.436170	0.872340	0.515474
12	GaussianNB	0.978228	0.099803	0.808511	0.053184	0.150564

As we can see from above image, ExtraTreeClassifier, XGBClassifier and Random Forest have scored the same on test data set.

Suggestion is to go for XGBClassifer as it was the fastest model.

6. Reflection

This project can be broken down into phases based on my encounter;

I. Getting the dataset - this was the easiest phase for me as the dimension of the data had already been reduced into Principal Components.

II. Exploratory data analysis - much wasn't done concerning this due to the fact that most of the features were principal components of the original data.

III. Model Selection – Choosing out of four algorithms was a bit confusing because of the resulting complexity of the Multi-Layer Perceptron and Decision Tree algorithm which gave an almost perfect score but then this might lead to overfitting.

Also I found the interpretation of this model to be difficult due to the anonymity of features which if was present, might explain the problem better. The most difficult phase for me was picking the right algorithm to use in building the final model.
Finally, embarking on this project has taught me important lessons about problems that may arise from solving a real life problem. Such lessons include, being able to work with data at an abstracted level, there is no perfect algorithm we can only find a very good one that suits the problem and sometimes the simple model is the best option!

7. Improvement

As stated earlier in the justification section, in the global context of credit card fraud detection this model might not be a silver bullet but it is definitely a great starting point to a larger project that seeks to tackle credit card fraud detection globally.

Improvement on solving the problem of credit card fraud detection would be industry/environment specific but some general rules apply, like a proper and detailed exploratory data analysis of variables that might explain the outcome of a transaction.
Also more data could be collected, which would put the complexity and strength of advanced deep learning techniques to good use in building a more robust and accurate algorithm.