Machine Learning Engineer Nanodegree

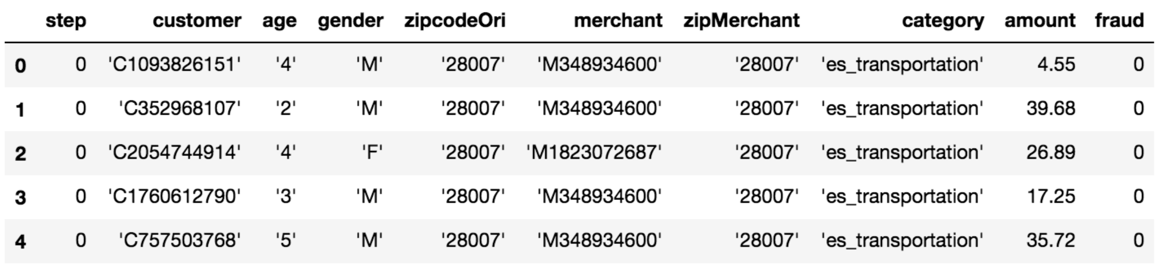
Capstone Project

Timi Ajiboye
September 17, 2017

I. Definition

Project Overview
As payment technology gets better, there are increasingly new ways to commit fraud. Although prevention is the best way to mitigate fraud, fraudulent agents are adaptive, and given enough time, usually find ways to bypass these measures. Techniques that allow for generalising the detection of fraud are important in the quest to catch all fraudsters once (algorithmic) prevention has failed. Several techniques have been developed to detect fraudulent transactions such as neural networks, genetic algorithms, data mining, clustering techniques, decision tree, Bayesian networks etc. In this project we will train a classifier to predict fraudulent transactions with synthetic transaction data generated by the BankSim payments simulator

Problem Statement
The goal is to train a suitable model that can generalize well enough to predicting fraudulent transactions given the least amount of input variables possible.



	step	customer	age	gender	zipcodeOri	merchant	zipMerchant	category	amount	fraud
0	0	'C1093826151'	'4'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	4.55	0
1	0	'C352968107'	'2'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	39.68	0
2	0	'C2054744914'	'4'	'F'	'28007'	'M1823072687'	'28007'	'es_transportation'	26.89	0
3	0	'C1760612790'	'3'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	17.25	0
4	0	'C757503768'	'5'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	35.72	0

The diagram above shows the (10) input variables we have in the dataset. To increase the efficiency of the model we're trying to train, it's important that we remove features that do not have any effect on the value.

However, the major problem with payment/transaction data is that it's often highly unbalanced; there are usually way more safe transactions than fraudulent ones.
As a result of this, the model may start to classify fraudulent transactions as safe since a huge majority of it's training was done with safe transactions.

Metrics
In this project, we will be using the following to measure the performance of the model:

- **Accuracy:** Accuracy is one of the simplest metrics as it's just the number of correct predictions made divided by the total number of predictions made, multiplied by 100 to turn it into a percentage. Ordinarily, accuracy would be a poor metric for such an unbalanced dataset. However, this imbalance is to be handled, so accuracy will work fine. Essentially, accuracy tells us the ratio of how many outputs our model predicted right relative to how many there are in total.
- **Precision:** Precision is the number of True Positives divided by the number of True Positives and False Positives. It is the number of positive predictions divided by the total number of

positive class values predicted. It is also called the Positive Predictive Value (PPV).

- **Recall**: Recall is the number of True Positives divided by the number of True Positives and the number of False Negatives. It is the number of positive predictions divided by the number of positive class values in the test data. It is also called Sensitivity or the True Positive Rate. Recall can be thought of as a measure of a classifiers completeness. A low recall indicates many False Negatives.
- **Training Time**: This is the time it takes a classifier to fit to the training data. As expected, more time taken is undesirable. In real world use, relatively few classification algorithms allow for new training data points to be added on the fly, most of the time, the model needs to be retrained on the entire dataset (including the new data points). This is why a low training time is a good thing.

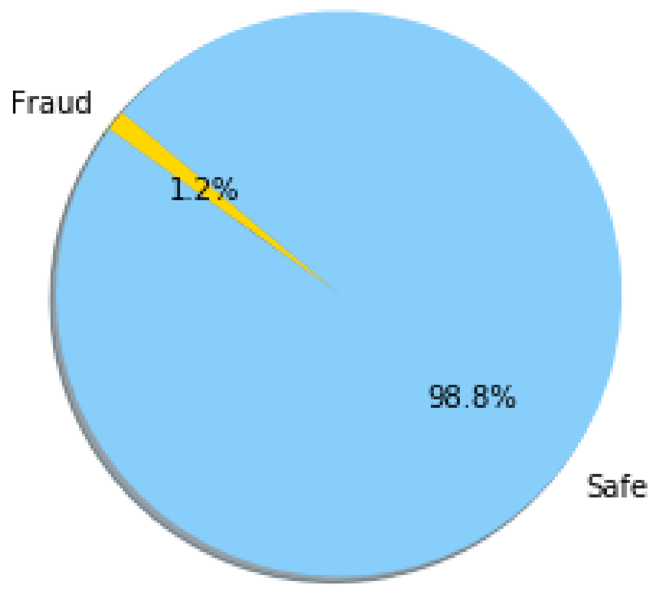
II. Analysis

Data Exploration

The data set has a total of 10 variables; 9 input variables and one output variable.
Out of these 9 input variables, 7 have String values - this is an issue for classifiers like Random Forests, and the features will require some form of encoding.

Furthermore:

- Total number of payments: 594,643
- Total number of fraudulent payments: 7200
- Total number of safe payments: 587,443
- Percentage of fraudulent payments: 1.21%
- Percentage of safe payments: 98.79%



Fraudulent transactions distribution

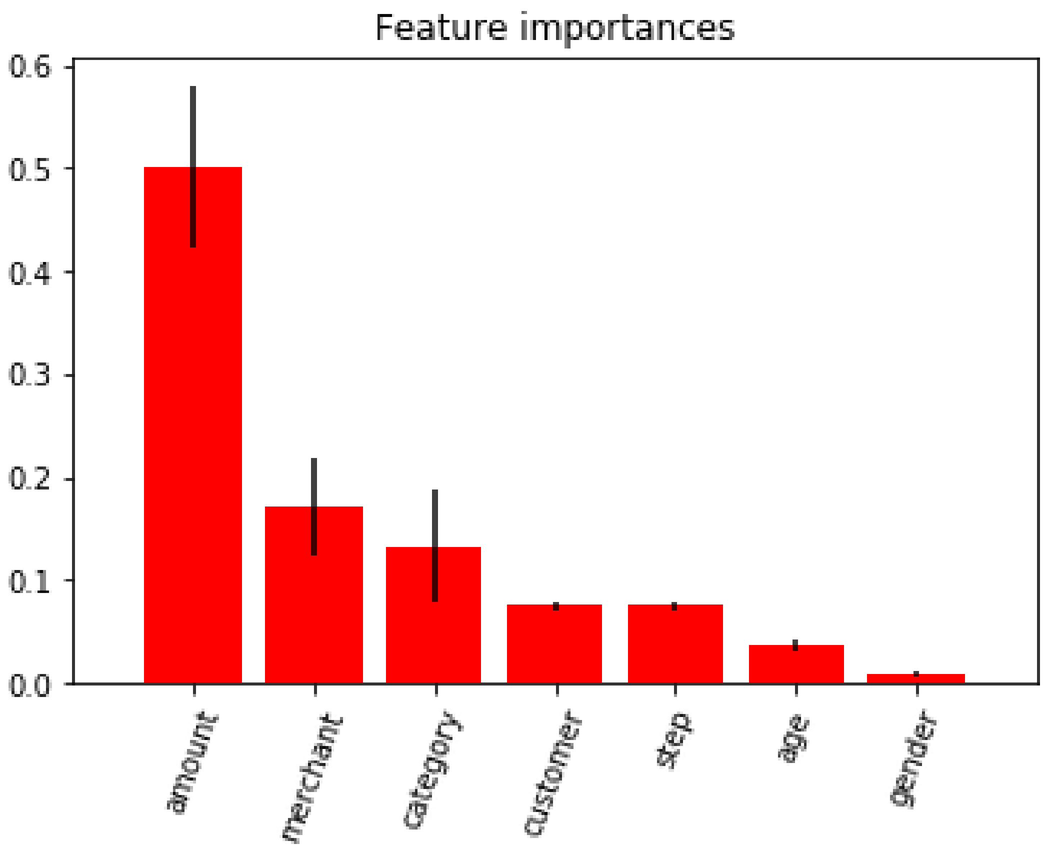
The above statistics confirm earlier suspicions of a highly unbalanced dataset.

It's also worth noting that there are are no features with NULL or NAN values and this is desirable as that sanitization step can be skipped.

Finally, it can be seen that two input variables (`zipcodeOri` and `zipMerchant`) only ever have just one value.

Exploratory Visualization

A simple bar chart of feature importances and inter-trees variability is plotted. The red bars are the importances of each feature and the black lines are the variabilities.



Feature importances

We can see from the bar plot above which features actually have significant effect on the target variable.

At this point, we take a deeper look at some of the most important features; `amount`, `merchant` & `category`.

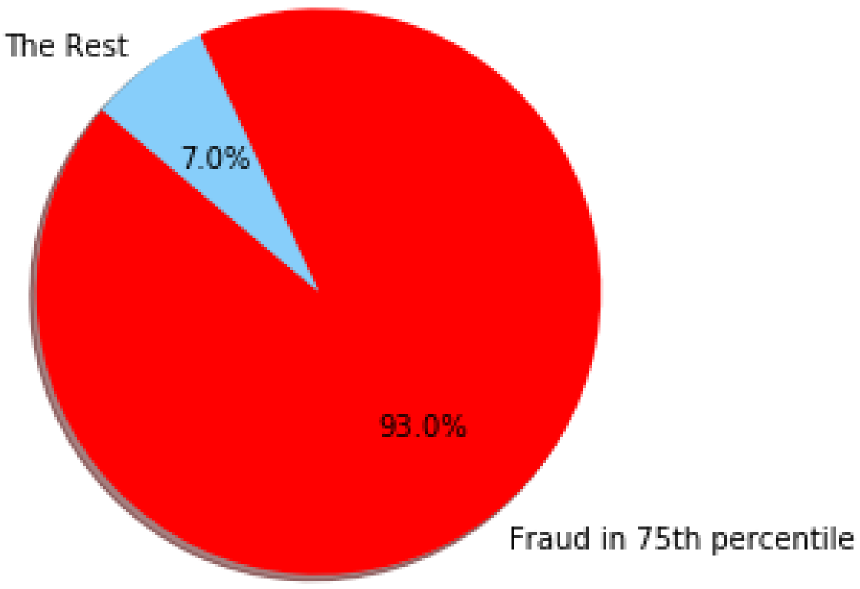
First and foremost, the most importance feature; `amount`. Here are some descriptive statistics about this column.

Stat	Value
Mean	37.890135
Standard Deviation	111.402831
Minumum	0.000000
Maximum	8329.960000

25th Percentile	13.740000
50th Percentile	26.900000
75th Percentile	42.540000

From the above table, specifically the standard deviation, we can see that amount varies quite a lot.

Furthermore, an amount of 42.54 is greater than 75% of the amounts in the table but the maximum amount is 8329.96, which is a relatively huge gap. Because of this, we're going to look at the distribution of fraudulent transactions that exist when the amount is equal to or above the 75th percentile.



Fraudulent transactons in 75th percentile

The fact that 93% of the amount of fraudulent transactions exists where amount is greater than 75% of the other amounts shows just why the feature is very important in prediction whether a transaction is fraudulent or not. It means that that the higher the amount is, the higher the chances of the transaction being fraudulent.

Finally, there are a total of 50 merchants and 15 categories.

Algorithms and Techniques
Again, the major problem we have with this dataset is how highly unbalanced it is. Thankfully, this is a relatively easy problem to solve with sklearn's implementation of certain classifier algorithms. These classifiers allow one to specify `class_weight` as `balanced`.

Balanced Class Weight: This mode essentially implicitly replicates the smaller class until you have as many samples as in the larger one (and as such, balancing the dataset).

Some of the classifiers that accept this `class_weight` parameter are:

- Logistic Regression
- Random Forests Classification

Logistic Regression: Logistic Regression is a type of classification algorithm involving a linear discriminant. Unlike actual regression, logistic regression does not try to predict the value of a numeric variable given a set of inputs. Instead, the output is a probability that the given input point belongs to a certain class, we selected this algorithm because we are dealing with a binary classification problem.

Random Forest Classifier: According to this paper and other research journals, it is said that this classifier works perfectly on our type of datasets especially with the fact that it’s training time is faster in terms of performance optimization.

It builds multiple such decision tree and amalgamate them together to get a more accurate and stable prediction. This is direct consequence of the fact that by maximum voting from a panel of independent judges, we get the final prediction better than the best judge.

Benchmark

To benchmark our models, we used a Gaussian Naive Bayes Classifier to make predictions on our initial datasets. The Gaussian NB Classifier was trained on 66% of the data and tested on the remaining 33%.

Here are the scores that our final model will need to beat:

- **Accuracy:** 97.2%
- **Precision:** 63.9%
- **Recall:** 90.3%

III. Methodology

Data Preprocessing

Feature Transformation: Due to the fact that a lot of the input feature columns have their values as strings, **Label Encoder** was implemented to change these to integers (out of a finite list of integers for each feature).

Feature Selection: This was done in two stages:

1. Two features (`zipcodeOri` and `zipMerchant`) had only one outcome throughout the entire dataset. This means that they do not have any effect whatsoever on the target variable and as such removed.
2. Using an **Extra Trees Classifier**, we were able to visualize how important or unimportant certain features are. As a result of this,

the `age` & `gender` features were removed from the dataset due to their low importance and inter-tee variability.

Implementation
In order to choose between these two algorithms, **K-Fold Cross Validation** using average/mean for *accuracy, precision, recall and training time* for scoring was employed.

Algorithm	avg. **Accuracy**	avg. **Precision**	avg. **Recall**	avg. **Time**
Random Forest Classifier	0.995643	0.947197	0.863266	76.114177
Logistic Regression	0.948668	0.587334	0.914796	2.163360

The **Random Forest Classifier** was selected as the final model due to it's high accuracy of 99.6%, precision of 94.7% and though it has a lower recall than **Logistic Regression's**, 86.3% is still pretty high.

Refinement
The training time for the Random Forest Classifier at this point left a lot to be desired and as such **Grid Search Cross Validation** is used to tune the parameters to attain a shorter training time without reducing the other metrics dramatically.

I suspected that the number of estimators (250) was what was responsible for the high training time, hence, we're going to try lower values (along with varying values of `max_features`) of the **RFC.**

	Tuning for best accuracy	Tuning for best precison	Tuning for best recall
Best features	max_features: auto, n_estimators: 50	max_features: auto, n_estimators: 10	max_features: auto, n_estimators: 45
Score	0.995683	0.949439	0.865686

All metrics perform better when `max_features` is on `auto`. Because of this, I set n_estimators as 10 as it yields the highest precision and shortest training time.

IV. Results

Model Evaluation and Validation

To verify the robustness of the model, I trained it on 20% of the data and tested it on the remaining 80%. This is meant to give an idea as to how the classifier generalizes to unseen data - If it's trained with a much smaller amount of data, tested on data that's x4 of the training data size and it still manages to perform well, it means that it works ideally on never before seen data.

The score(s) of the prediction on the testing set are as follows:

- **Accuracy:** 99.5%
- **Precision**: 94.3%
- **Recall:** 85.8%
- **Training time:** 0.51 seconds.

Justification

Compared to the results in the **Benchmark** section, there has been a small uptick in recall while training time as taken a huge dip.

Accuracy and precision are more or less the same but this isn't a problem as they were already quite high in the first place.

	Benchmark scores	**Post refinement scores (trained on 20% of data)**
Accuracy	97.3%	99.5%
Precision	63.2%	94.3%
Recall	90.3%	85.8%

From the above table, the model does better in two metrics (accuracy and precision) but does worse in recall.

Ideally, one would expect the final model to outperform the benchmark in every regard. However, it can be proven that the final model is still a better alternative to the benchmark's model by using the **f1 score.**

F1 Score: This is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. It is calculated as:

$$2 * (Recall * Precision)/(Recall + Precision)$$

- **Benchmark F1 Score:** 70.2%
- **Final Model F1 Score:** 89.3%

It can be seen that despite the slightly lower recall, the final model outperforms the benchmark's model.

The main problem with the dataset and hence our goals is that it's highly imbalanced, where we have <1.5% of transactions as fraudulent. Given data skewness, classifiers will tend to prefer normal (negative class) transactions and will have challenge identifying fraud positive classes.

Hence to overcome this, the algorithms we chose to explore had to be able handle this implicitly, by replicating the smaller (positive class) enough to eliminate this imbalance.

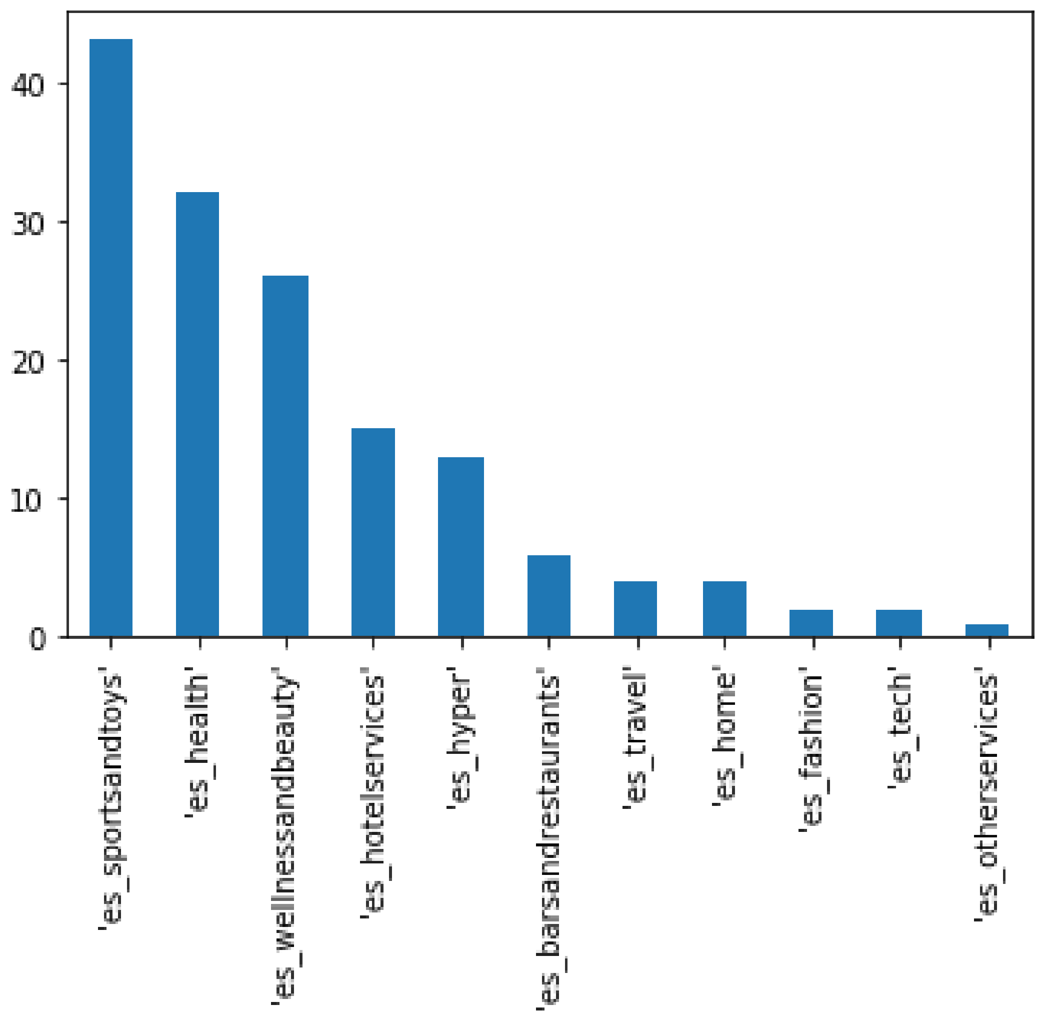
Because of this algorithm(s) choice we were able to relatively easily solve the imbalance problem and still end up with techniques that are usually good with this type of problem (fraud detection).

As can be seen from the table above, the scores that were arrived at are quite good and as such, satisfactory.

V. Conclusion

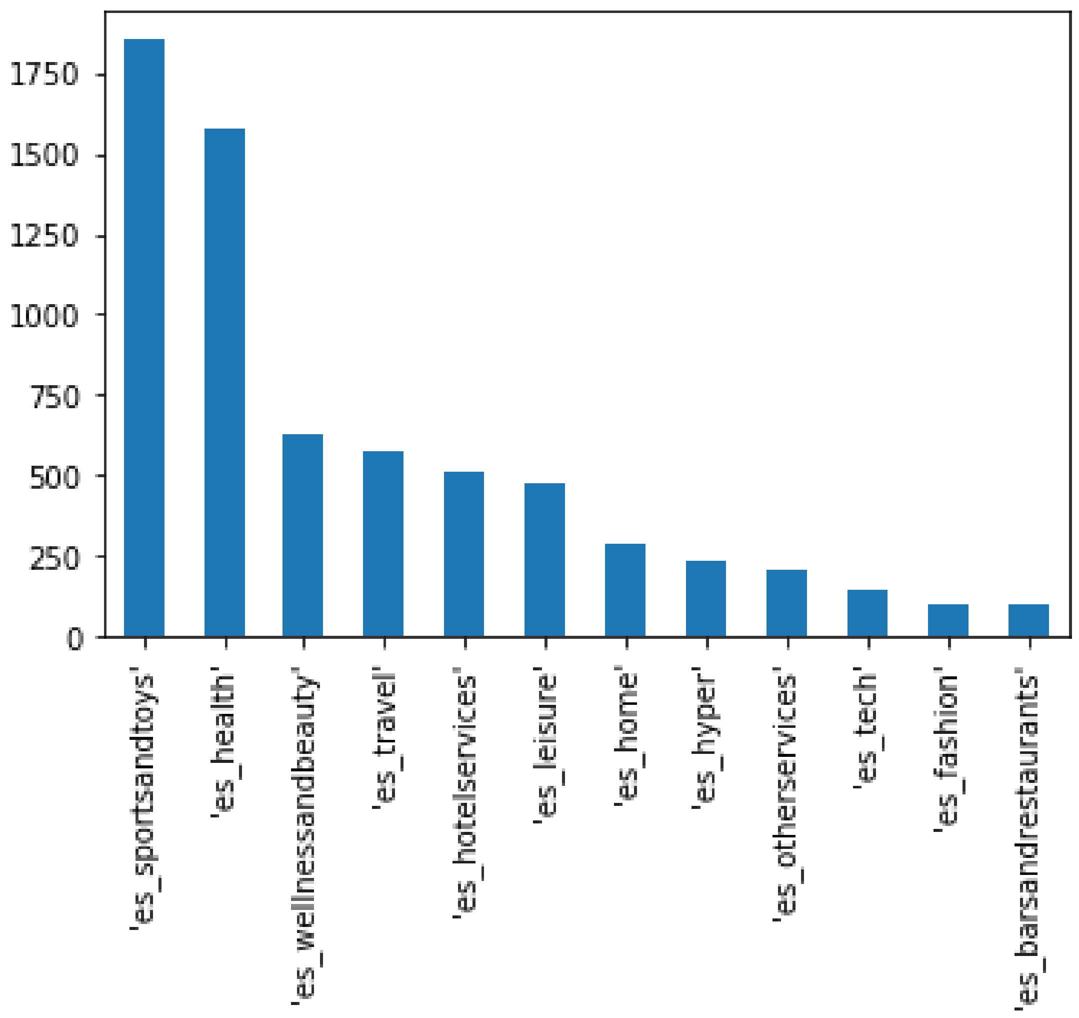
Free-Form Visualization

I decided to take a look at the fraudulent transactions that exist when the amount is equal to or less than the 25th percentile and see the distribution of fraudulent transactions across categories.



Distribution of fraudulent transactions across cheap (<= 25%) purchases

Found it very interesting to note the categories that had the highest count of fraud when they're cheap; Sports & Toys, Health, Wellness& Beauty. I then decided to compare it to the distribution of fraudulent transactions across categories for transactions with amounts greater than or equal to the 75th percentile.



Distribution of fraudulent transactions across expensive (>= 75%) purchases

For the categories with majority of the fraudulent transactions, it's an almost identical ranking. This leads me to believe that it might just be easier to be fraudulent with certain categories of products/services regardless of how much it costs.

Reflection

In summary, these are the steps that make up the process for completing this project:

1. Find an interesting, practical problem and an accompanying public dataset.
2. Download and pre-process the data; feature selection, feature transformation; encoding.
3. Identify the best metrics to benchmark our classifier.

4. Choose algorithms that are good at dealing with imbalanced data.
5. Implement the classifier that comes out on top during benchmarking.
6. Refine the classifier using hyper-tuning.

Step 4 was a bit of a challenge to arrive at; there are so many ways to handle imbalanced dataset but I had my sights set on something simple to use and out of the box. I had a hunch that such a solution had to exist because imbalanced datasets seem like such a common problem that should have it's solution already abstracted away to the inner workings of the classification algorithm.

Improvement

I suspect that there might be ways to improve this, here are some options that are worthy of exploration:

- Using the SMOTE technique to generate more data as opposed to using the balanced class weight method.
- Furthermore, if the above were to be done, using Stratified ShuffleSplit for cross validation might yield improvements.
- It's also possible that outlier detection algorithms like EllipticEnvelope, IsolationForest and LocalOutlierFactor might work better.