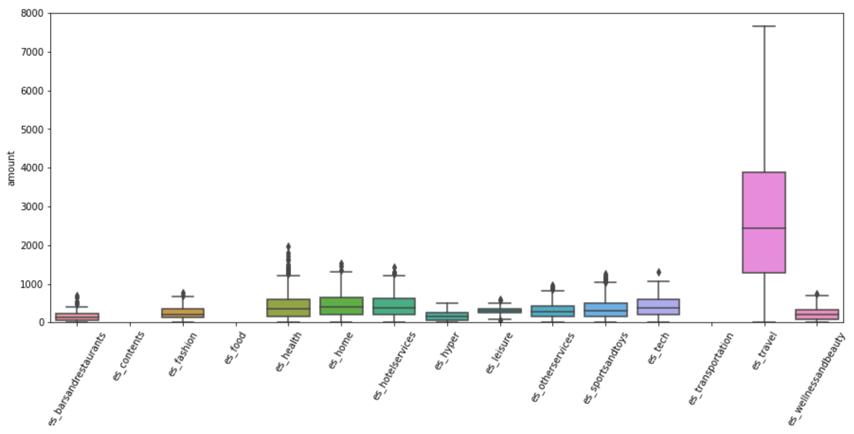
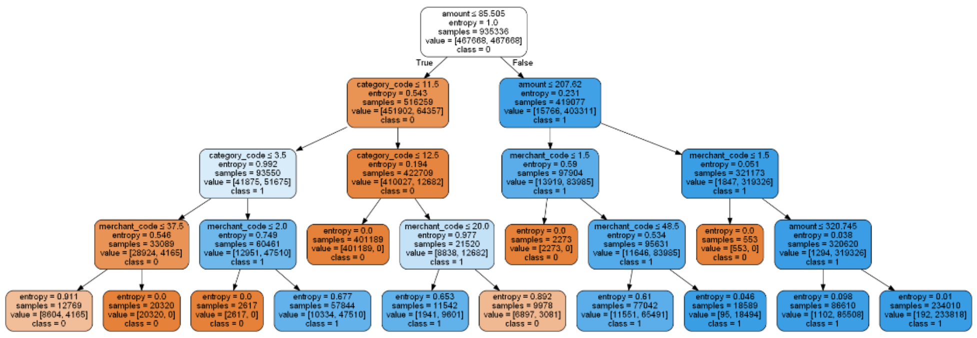
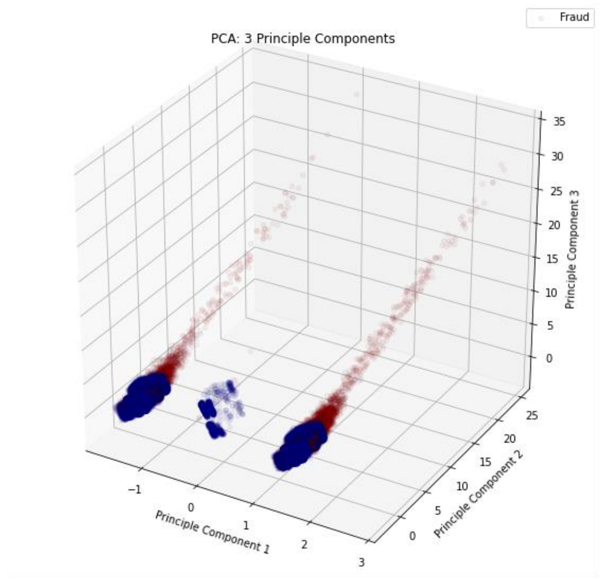






Fraud Detection

CLASSIFICATION MODELS: KEY CONSIDERATIONS AND IMPLEMENTATION

Roderick Cox | DSA 8023 | 04/04/2020

Project Outline

A bank wishes to create and implement a machine learning model to detect instances of online credit card fraud. The stakeholders of the project, who are not data science experts, have outlined 3 objectives to be fulfilled by the project:

1. Create an effective model for detecting instances of online credit card fraud using the dataset provided.
2. Identify which factors indicate whether a transaction is fraudulent.
3. Describe how a model could be implemented into the bank's strategy to combat fraud.

The second requirement contains the implication that whichever model is used should be interpretable, so that the bank understands how and why the model classifies transactions.

This report will detail the process and results of the project, evaluating the approach that was most effective at meeting the three objectives.

Section 1: Exploratory Data Analysis (EDA)

Variable Definitions

The dataset provided by the bank consisted of 591,746 observations with 10 variables. The bank did not provide any definitions of the variables, and while most were self-explanatory such as *amount* being the value of the transaction, others were more ambiguous or contain ambiguous values. Loading the dataset and inspecting the names and values that each variable took lead to the following assumed definitions.

- *step* – the index of transactions recorded per customer.
- *customer* – unique identifier for each customer.
- *age* – the age of the user whose account registered the transaction.
- *gender* – the gender of the user whose account registered the transaction.
- *zipcodeor* – the zip code where the transaction was registered.
- *merchant* – the merchant who received the transaction.
- *zipMerchant* – the zip code of the merchant who received the transaction.
- *category* – the category of good/service for which the transaction was made.
- *amount* – the value of the transaction.
- *fraud* – target variable which indicates whether the transaction was fraudulent.

Variable Values

A number of issues were identified when inspecting the possible values of each variable and the distributions of values that they took. Addressing these issues was necessary in building a viable machine learning model. Firstly, the variable *age* did not contain an integer value that corresponded with a user's age in years, but rather the values 0 to 6. It is common to subset ages into ordinal brackets, and it was assumed that this was the case. Machine learning models that treat *age* as a continuous variable will be able to effectively use a numeric feature that has been grouped in this way, even if some accuracy is lost by the exact age not being given, though this assumes that the age brackets were logically in order of size, and age brackets were of similar ranges. Secondly, the *gender* and *age* variables both contained "U" values and *gender* also contained a value "E". Further analysis revealed that a transaction contained an "E" for *gender* if and only if it also contained a "U" for *age*. There were sufficient instances of both occurrences (1173) to make it extremely unlikely that this had occurred by chance. "U" is a common placeholder for "Unknown", and the occurrence of "U" for *age* and "E" for *gender* together indicated that these transactions were related to a user for which a gender and age were not applicable, such as a company or group. It seems likely that "E" signifies "entity" or "enterprise". The remaining "U" values for *gender* would then encompass people who chose not to identify as Male or Female ("M", "F") or those whose data is genuinely missing, or a mix of both. It was therefore concluded that "U" signified missing or otherwise not applicable, and "E" signified an entity. Thirdly, *zipcodeor* and *zipMerchant* were found to be redundant variables, as every observation took the same value and the columns were duplicates of each other. This is resolved in section 2.

Variable Distributions

Viewing the distributions of each variable is important for gaining understanding of the variables that are likely to impact a machine learning model, and more generally understand values that are indicative of online credit card fraud. They are also useful in revealing any abnormalities in the data that might negatively impact a machine learning model, e.g. too few data points for a particular category, or large expected counts of a certain value could indicate a fault in the collection of the data. Finally, exploring and understanding the data may reveal opportunities for feature engineering.

Firstly, the number of fraudulent transactions in the dataset was much lower than those which were non-fraudulent. Fraudulent transactions made up about 1.2% of the data.

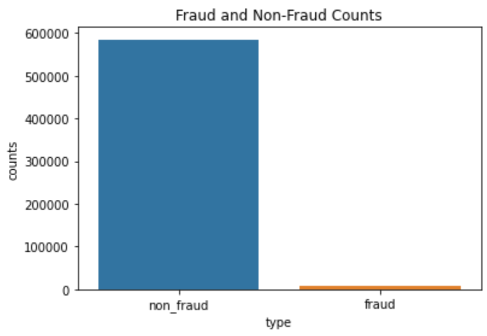


Figure 1: Unbalanced Data

Splitting the data into a data frame for fraudulent transaction and a data frame for non-fraudulent transactions allowed for effective comparison through visualization.

The first observation was that fraudulent transactions were spread across many categories, with sports, toys and health being particularly prominent.

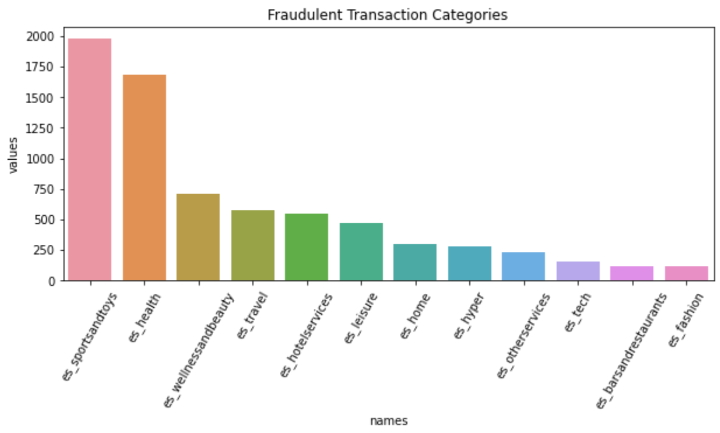


Figure 2: Categories - Fraud Data

By contrast, non-fraudulent transactions were dominated by transportation.

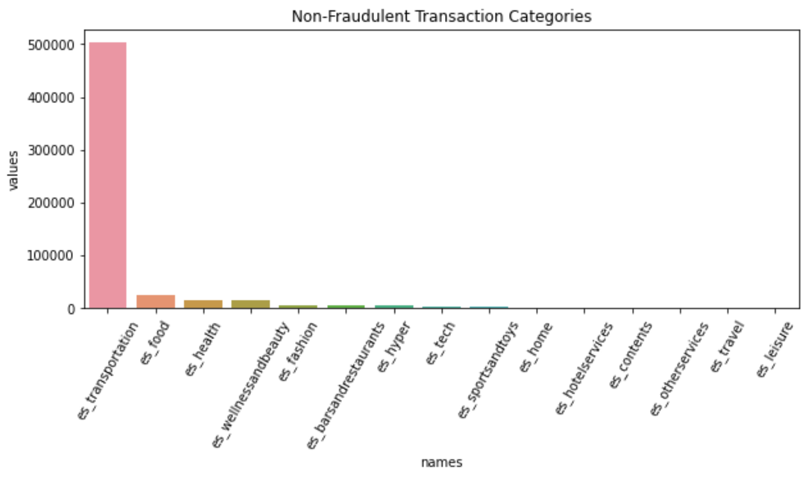


Figure 3: Categories - Non-Fraud Data

The distributions of *amount* were also notable. Both the distributions of the fraudulent and non-fraudulent *amount* were strongly positively skewed, but fraudulent transactions had much higher mean value. There were still, however, a significant proportion of fraudulent transactions with a low values.

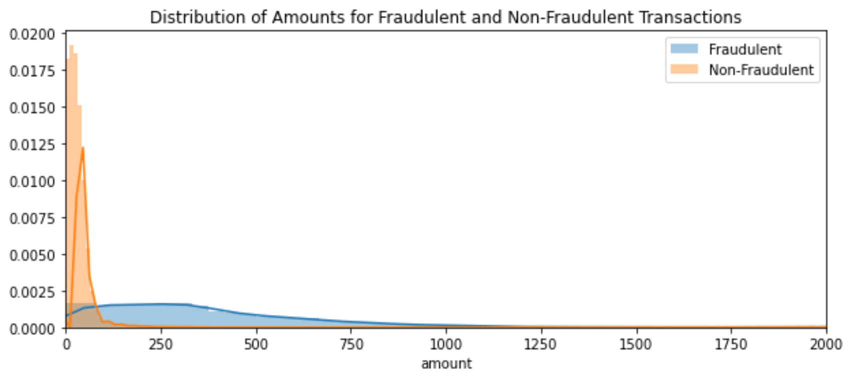


Figure 4: Amount: Fraud and Non-Fraud

Comparing the distribution of *amount* by *category* for fraudulent and non-fraudulent data revealed that the increase in *amount* for fraudulent data was not even across categories. In particular, travel was disproportionally higher amongst fraudulent transactions, while bars and restaurants saw a much lower increase. This particular insight will be explored further in section 2 and utilized in section 4.

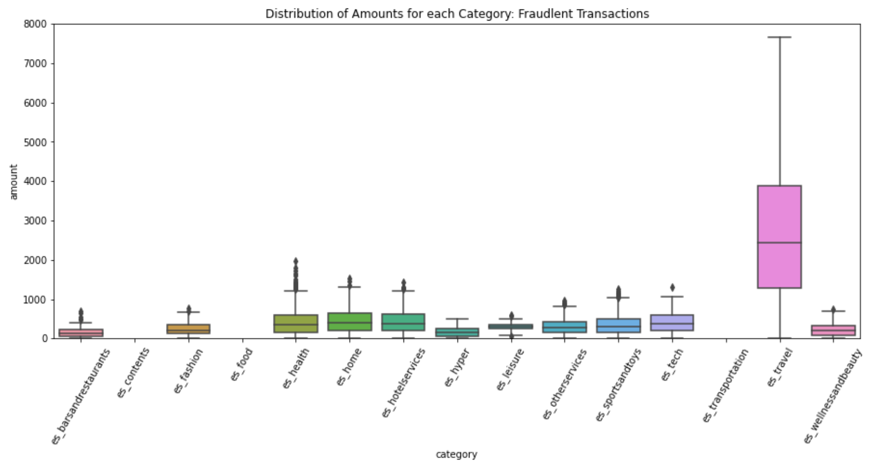


Figure 5: Amount by Category – Fraud

It should also be noted that some categories contained no fraudulent transactions. This was a limitation of the data, it is of course possible for fraudulent transactions relating to food and transportation, though since the data contained no fraudulent data on these categories, if a model learns to never classify such categories as fraud, this detrimental aspect of the model will not be reflected in the model's accuracy.

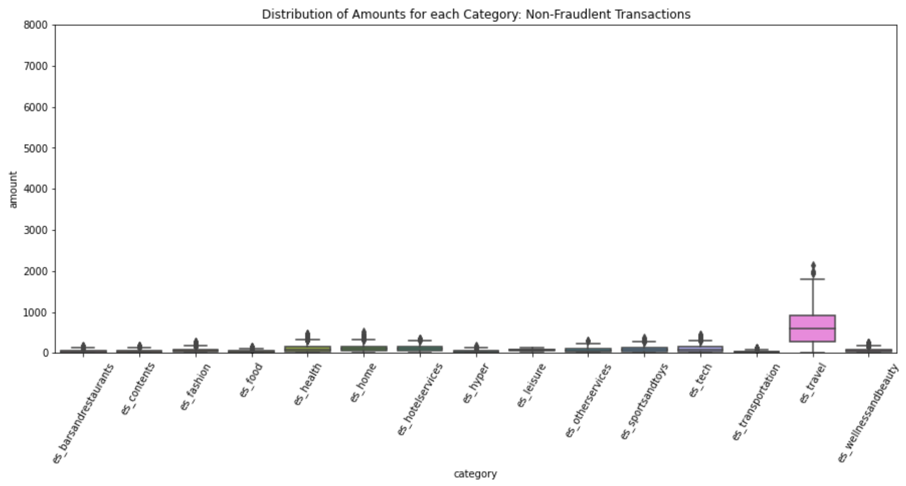


Figure 6: Amount by Category – Non-Fraud

A similar conclusion can be drawn from the *amount* by *merchant* for fraud and non-fraud, where some merchants saw a disproportionate increase for fraudulent transactions, and some where there was no data available. There were also some outliers in both the fraudulent and non-fraudulent plots, such as for "M62898500" where there was a significantly varied range of transaction amounts for fraudulent transactions, and some unusually high non-fraudulent transaction amounts, which would likely have made classifying these transactions quite difficult.

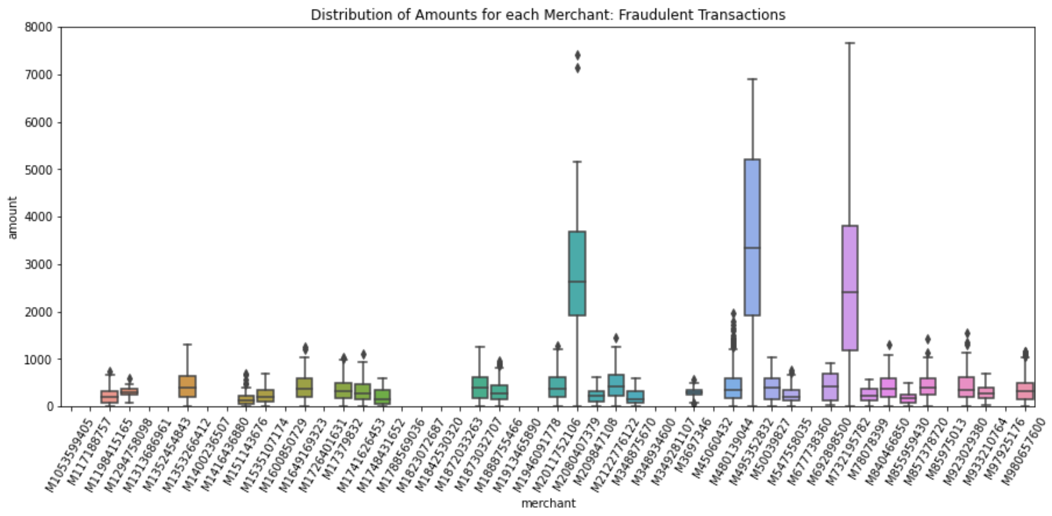


Figure 7: Amount by Merchant – Fraud

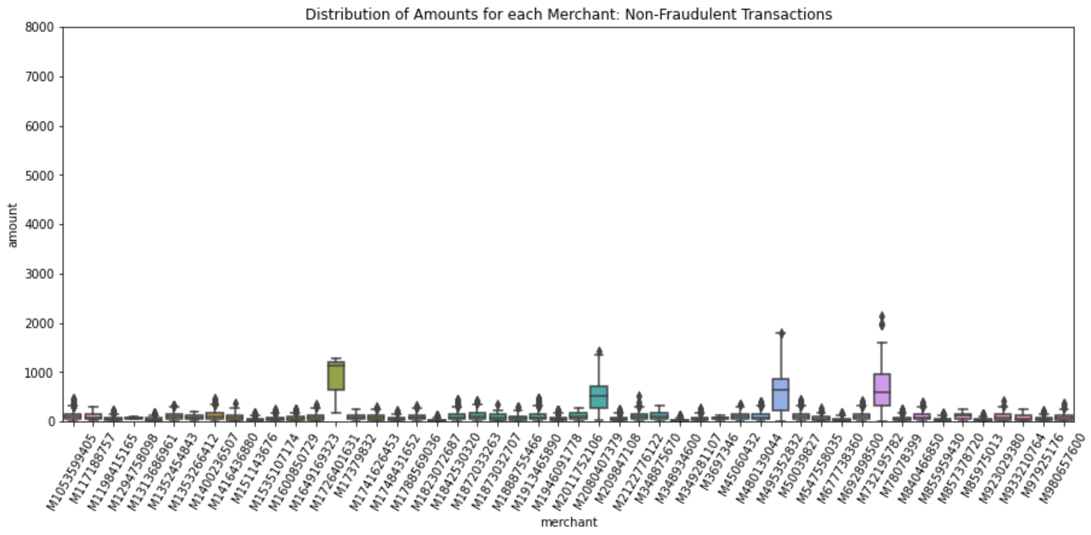


Figure 8: Amount by Category – Non-Fraud

Section 2: Preprocessing

Prior to building any machine learning models, a number of decisions needed to be taken with regards to processing the data.

One-Hot-Encoding

Many machine learning classification models expect to train and test on features that contain numerical data and are not equipped to process categorical data. As such, it was important to convert categorical data to numerical prior to using a model. A number of variables in the data were categorical, and each category would ideally be treated as an individual feature, with its own multi-faceted relationships with other features and the target variable. A common solution is to one-hot encode each variable, which is to create new columns for the unique values of these categorical variables, assigning the values "1" or "0" to indicate the presence of a value (n – 1 categories is required, as for one of the variables, the absence of all other categories implies the presence of the other). For the *gender* column, which was reduced to 4 categories “M”, “F”, “E” and “U”, one-hot encoding was a viable solution, however the *category* and *merchant* features had significantly more unique values, and one-hot encoding would have greatly increased the feature space, resulting in the issues caused by the “curse of dimensionality”. As the dimensionality grows, significantly more data points are required required to fill the feature space. If insufficient data is used, models are likely to overfit, that is, classify based on noise in the data rather than on meaningful relationships. It would also result in a large increase in the computational expense of training the models as the dataset grows. As such, the *gender* column was one-hot encoded, but the *category* and *merchant* variables were not.

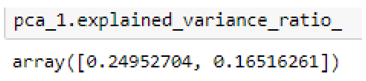
Label Encoding

Another way to handle the categorical data in the *merchant* and *category* features was Label Encoding. Label encoding assigns an integer value to each unique value of a feature and replaces the original values with the integer label. Label encoding is a simple way to deal with categorical data, but there are significant drawbacks to this approach. Machine learning models will interpret the feature values as ordinal values with some scale, which does not make sense for categorical values. For example, if type “C” is labelled “3” and type “F” is labelled “6”, the value “F” might be interpreted is twice the value of type “C”. This leads to problems for various models. A random forest will create rules using operators such as “greater than” or “less than” for the label encoded values, and clustering models will compute distances based on their values. These computations do not make sense for categorical values and will affect the outcomes of classification. When the number of labels is small, a model may still be able to derive meaningful relationships from the individual values but, as more labels are added, the individual values contribute less and their relative position on the scale of values will impact models more. A superior solution was utilized in section 3, however, for the initial modelling, the suboptimal solution of label-encoding was used for the *merchant* and *category* features.

Principle Component Analysis (PCA)

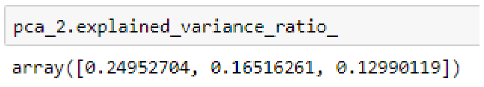
Some datasets require the use of dimensionality reduction techniques for machine learning models to be effective. Datasets that contain large numbers of features regularly suffer from overfitting as a result of the “curse of dimensionality” discussed previously. Principle Component Analysis (PCA) is a dimensionality reduction technique that extracts uncorrelated variables (principle components) from a feature set in such a way as to retain the maximum amount of variability of the data with as few principle components as possible.

PCA is a useful technique for datasets with large numbers of features, especially when many features are made redundant by being strongly correlated to others (multicollinearity). This dataset had a relatively small number of features, and since features were not strongly correlated, the variance explained was low when using 2 or 3 principle components.



```
pca_1.explained_variance_ratio_

array([0.24952704, 0.16516261])
```



```
pca_2.explained_variance_ratio_

array([0.24952704, 0.16516261, 0.12990119])
```

More than 6 principle components were required to explain 80% of the variability, which defeats the purpose of dimensionality reduction on a dataset that already contained few features.

Furthermore, after label encoding, the *merchant* and *category* variables take on arbitrary integer values that are unlikely to be correlated to any other variables, which limits the effectiveness of PCA to capture the dataset’s variability with a low number of components.

Lastly, in terms of interpretability, PCA has benefits and drawbacks. The benefit is that, using principle components, many variables can be visually represented in 2 or 3 dimensions if 2 or 3 principle components have been used. In classification problems, where the data is clearly linearly separable, this can reveal clusters of data. in this case, PCA was not as effective, and while some distinction could be seen between fraudulent and non-fraudulent data, there was too much overlap for there to be considered distinct clusters. The drawback is that replacing the original features with orthogonal transformations of variables composed using eigenvalue decomposition results in a loss of interpretability, especially for those who are not familiar with Principle Component Analysis. Ultimately, PCA for the reasons discussed, should not be applied to this data, though the results of the models after using PCA are available for comparison.

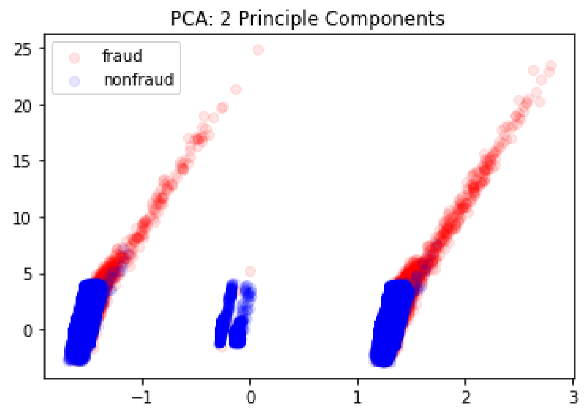


Figure 9: PCA - 2 Components

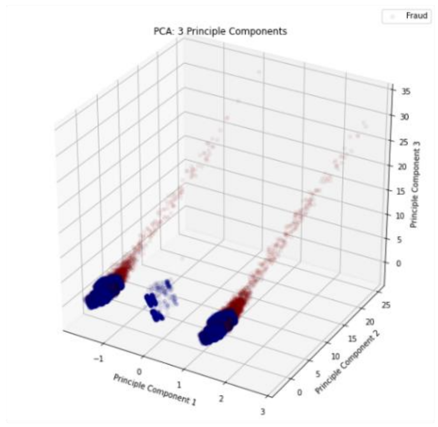


Figure 10: PCA - 3 Components

Features to Exclude

There were both performance and context related reasons to exclude some features from the dataset before modelling. As was previously discussed, label encoding was likely to negatively impact the *merchant* and *category* variables, and in section 4 these were replaced with features that resulted in higher model performance. The *zipcodeor* and *zipMerchant* features provided no information and were removed. However, it should be noted that variables that concern location are likely to be strong indicators of fraudulent transactions, particularly in relation to a customer's usual locations. This will be discussed further in section 5. Finally, the *customer* variable was also excluded from the model for contextual reasons. The model may be able to use the *customer* values effectively to classify fraud, but the objective is to create a model that can generalize to other bases of customers, and so should not rely on specific values of *customer* for the classification task.

Unbalanced Data

The balance of the data was an important factor to address prior to training models. The dataset provided showed a strong imbalance of fraudulent and non-fraudulent data, with fraudulent data making up 1.2% of the data. This presented two problems:

- There may be insufficient data for the model to learn the characteristics of a fraudulent transaction.
- Some models may learn to consistently classify transactions as non-fraudulent to achieve a high accuracy.

Aside from requesting more data, there is little that can be done to address the first issue, however, oversampling may be used to address the second. Oversampling is a method of artificially increasing the size of the training data by resampling the underrepresented class data with replacement. In this case, to attain an equal proportion of fraudulent and non-fraudulent transactions, the fraudulent data was resampled many times with replacement to match the number of non-fraudulent transactions Oversampling was crucially only used on the training data after the train-test split. Oversampling prior to the split would result in many exact replicates of the testing data being present in the training data, which would unfairly boost the performance of the model, and the validity of the model would be lost. Furthermore, oversampling was not also used on the testing data. It was necessary that the testing data was not contaminated by oversampling since it would have introduced further uncertainty about the model's performance. For example, if a certain type of fraudulent transaction was duplicated a disproportionate number of times, the model's accuracy may have been an overestimate or underestimate of the effectiveness of the model as it would have been repeatedly tested on the same kind of fraudulent transaction. After oversampling, the distribution of the training data is equal.

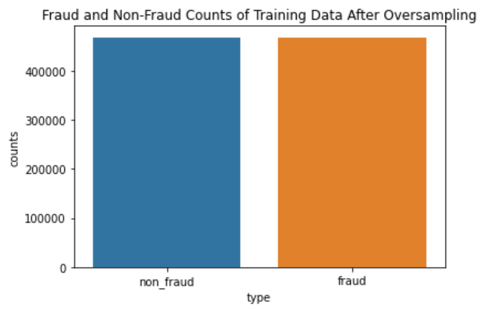


Figure 11: Data Balance after Oversampling

Section 3: Modelling

Approaches to the Modelling

There were four criteria for a desirable model in the context of this fraud detection task: accuracy, sensitivity, scalability and interpretability. Accuracy, which is the rate at which a model predicts the correct class, is the most obvious desirable quality of most models. However, in this task, more important than accuracy is sensitivity. Sensitivity is the rate at which the model correctly predicts positive cases – in this case, the rate at which the model correctly predicts fraudulent transactions. This is important as the model may achieve a very high accuracy by classifying non-fraudulent transactions very effectively, but still be quite poor at classifying the smaller set of fraudulent transactions. In this task, it was clearly more important for the model to correctly identify fraud than correctly identify non-fraud. It may be frustrating or costly for the bank to perform security checks on customers whose transactions are incorrectly flagged as fraudulent, but this would be much less significant than misidentifying a fraudulent transaction. There are other cases, such as spam filters, where the reverse is true, and the specificity (the rate at which a model successfully predicts negative cases) is more significant. A spam filter is designed to identify spam and remove it from an inbox, but it is more important to avoid removing a user's genuine mail.

The third factor, scalability, is particularly important for large datasets, and so is especially important if the bank wishes to maintain and grow the dataset over time and train models on the new data. There is also the related consideration of speed. In section 5, where implementation into the bank's strategy is discussed, the speed of classification is an important factor when considering online transactions. Many machine learning models suffer from exponential increases in computational expense when training on more data, so any model used should be relatively fast, and scale well on large datasets.

The final factor, interpretability, is a specific criterion for this task. The bank wishes to be able to understand how the model is using the features it has provided to classify transactions. This may be for their own understanding or some other practical purpose. The bank may wish to be able to provide a definitive answer to a customer who queries why one of their transactions was flagged as fraudulent. A comparable example is that of mortgage provision, where banks may use machine learning models to assess whether a customer is a suitable candidate for a mortgage. Many countries have legal requirements that a provider must be able to provide, when asked, an explanation for why a customer was rejected, partly to deter credit providers from rejecting candidates on the basis of ethnicity, gender or religious belief. Few machine learning models are considered to be "black boxes" in the sense that they learn without any user knowledge of how they learned, though some are undoubtedly more interpretable than others.

Possible Models

K-Nearest-Neighbors Classifier (KNN)

K-Nearest-Neighbors is an algorithm that classifies data based on proximity to other data points. After scaling all the features, the algorithm computes the distance from each point to every other point. The class with the largest number of the closest n points is the class assigned to the point. This model is very intuitive, using the logic that similar classes have similar values. The main drawback of KNN is that the ability to scale is very poor. Since the distance between each point and every other point must be computed, the computational expense grows exponentially with every additional data point.

Viewing the classes predicted by KNN by projecting the results onto 3 of the features used for classification, it is easy to see that KNN struggles to classify points that appear to lie far from the usual locations of classes.

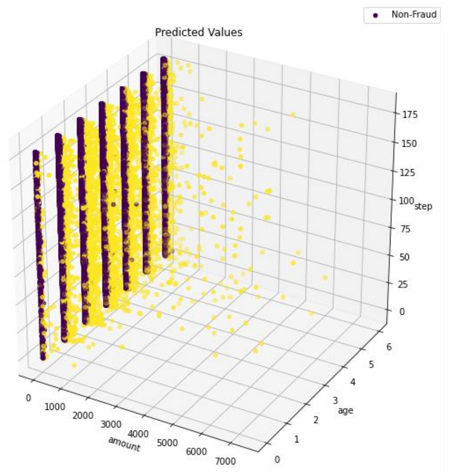


Figure 12: KNN Predicted Values Representation

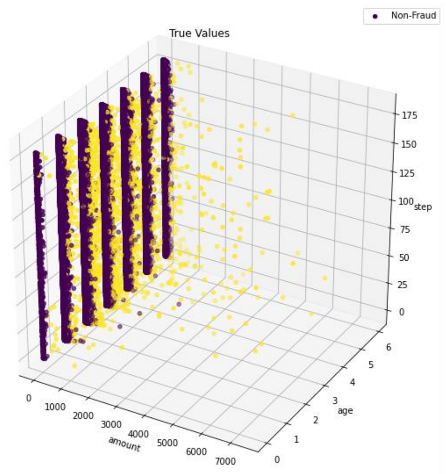


Figure 13: True Values Representation

Neural Network Classifier

A feed-forward linear neural network can also be used for classification. Neural networks, inspired by neurons in the brain, use a sequence of layers connected by weights to take an input vector (the values in the feature set), and output classes (fraud or non-fraud). To compute the weights when training the network, initial weights between layers are assigned completely at random. Training data is then passed through the network (forward-propagated) often in batches for the sake of speed, and the loss is computed, which is a measure of how the difference between the predicted and actual classifications. The model then performs backpropagation, which uses gradient descent to adjust the weights between layers so as to achieve a more accurate result.

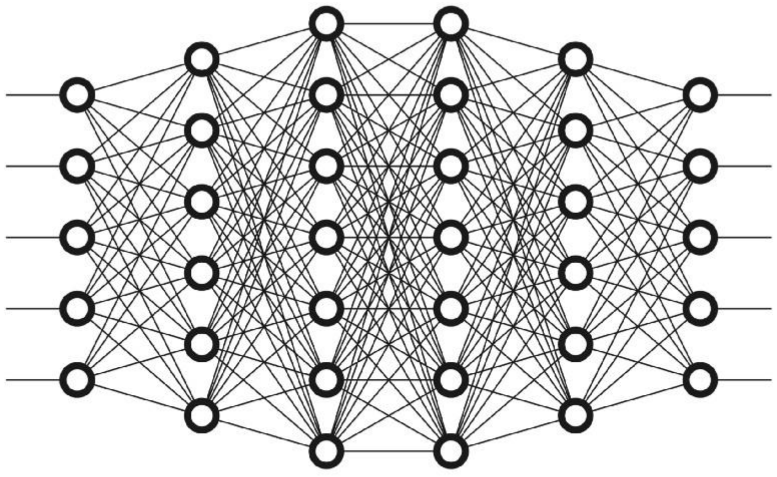


Figure 14: Neural Network Representation

Neural networks were limited in this case by the interpretability requirement. While these models can have excellent results, they are often criticized for being most like a "black box", in this sense that their results are difficult to explain or interpret. The user will not be able to detect patterns in the weights for a network of any real size, and will not be able to explain how the model chose the weights of the network, except by referring to the process of backpropagation. The number of layers to use in a neural network is not definitive, and size is often considered a hyperparameter. It is also hard to gauge whether a model has trained optimally using gradient descent or whether gradient descent has resulted in weights being stuck at a local minimum or being unable to descend further if the learning rate is too high.

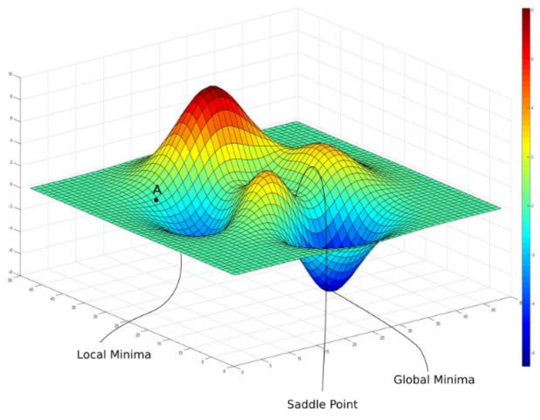


Figure 15: Gradient Descent Representation

The computational expense of the neural network depends on the number and size of the layers, however it scales linearly as the quantity of data increases, since backpropagation is performed after every batch of inputs.

Decision Tree

A decision tree, as the name suggests, builds a sequence of classification rules in a tree-like flow chart structure. The nodes of a decision tree are rules that split the data, such as "step > 50", and the branches are the binary results of the rule. Finally, the leaves of the tree are the resulting classes. Without going into excessive detail, the decision tree chooses the rules by calculating the "information gain" for each variable at each stage of the tree. Information gain is a measure of how well a given split in a variable's values separates the training data into the target classes. Information gain is calculated from the "entropy", which is a measure of homogeneity.

A basic decision tree, produced by limiting the max-depth parameter, allows for effective visualization. Decision trees are very simple to understand and excellent for visualization. A user can trace the process from node to node until they reach a leaf. Decision trees also scale very well on large datasets.

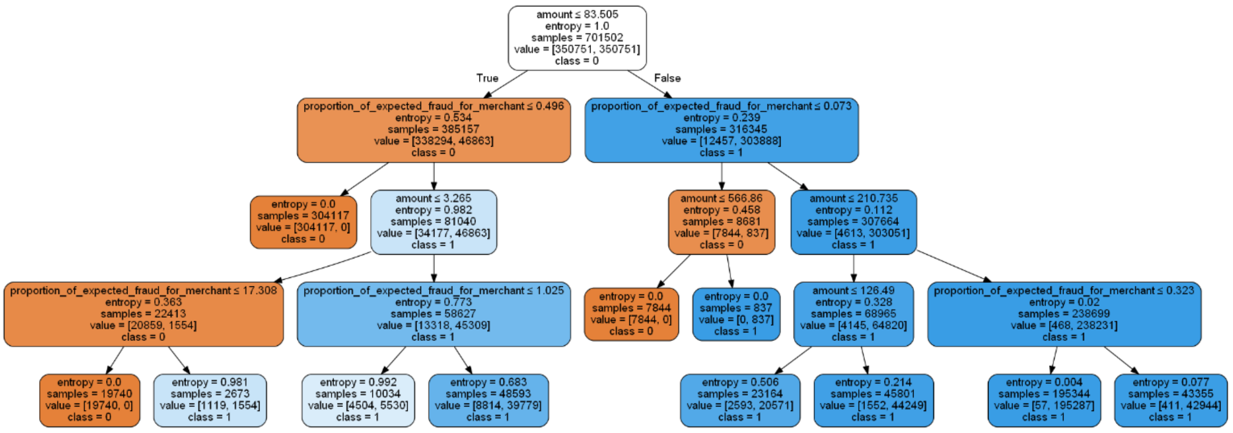


Figure 16: Decision Tree - Limited Depth

One common drawback however is the tendency of decision trees to overfit, though this issue is somewhat well addressed by instead using a Random Forest model.

Random Forest

Random forests are an extension to the decision tree algorithm. A random forest creates a number of decision trees from random samples of the data, and then classifies each data point by the outputting the mode class of the decision trees. In this way, the model corrects for overfitting, and tends to outperform individual decision trees.

The model is similarly intuitive, and individual trees in the random forest can be extracted and visualized as before. While there would be more trees to inspect, the core principles are the same with the extension of allowing the trees to vote on classes. The computational expense of a random forest is a linear function of the number of trees, so while larger it scales just as well.

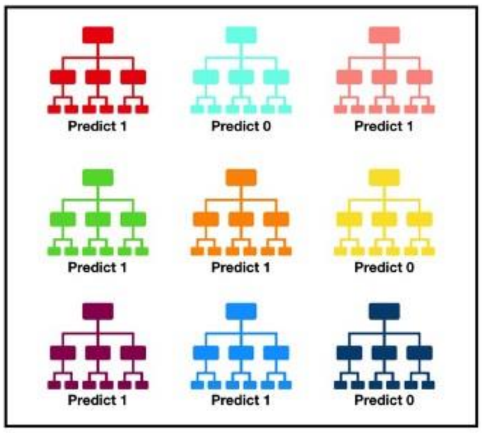


Figure 17: Random Forest Representation

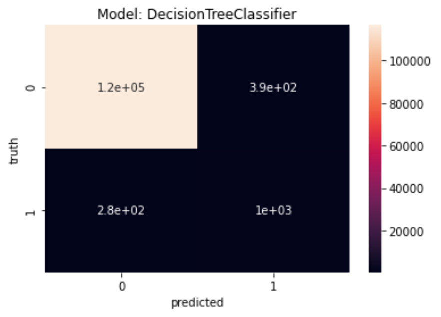
Comparison of Initial Results

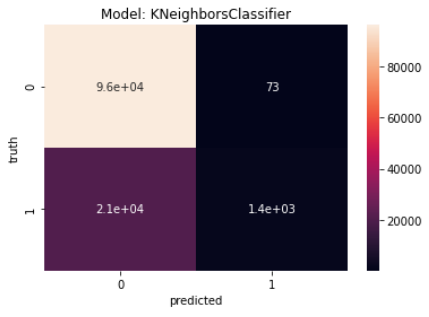
The results of the models prior to any feature engineering are shown below. The model results when using the PCA data is also given. Note that KNN and Sequential (Neural Network) features had been scaled prior to modelling.

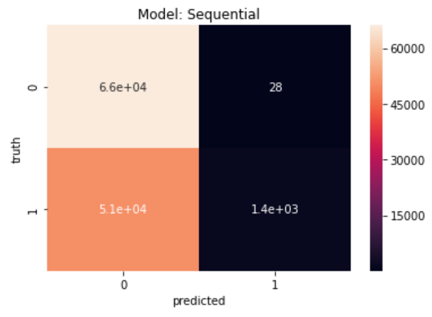
```
Accuracy: 0.9939416727573839
Sensitivity: 0.7549928673323824
Specificity: 0.996806259218880
```

```
Accuracy: 0.8447944435760185
Sensitivity: 0.06839784918834892
Specificity: 0.9990884137386178
```

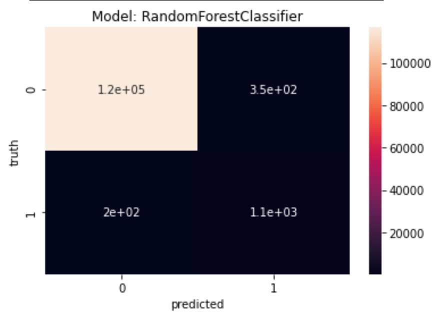
```
Accuracy: 0.5707055344317702
Sensitivity: 0.02690531399114654
Specificity: 0.9995768283283207
```







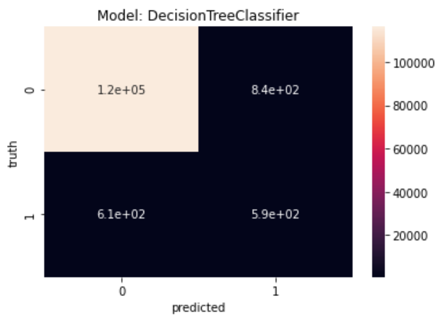
```
Accuracy: 0.9951753070355177
Sensitivity: 0.8249056603773585
Specificity: 0.9971031707035706
```



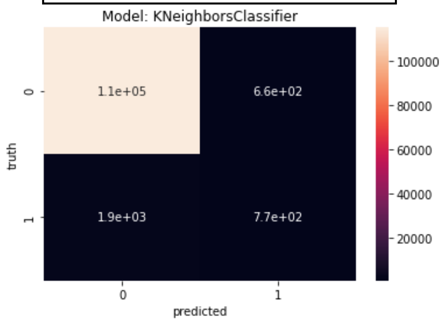
The best model is a random forest, trained on data which does not use the PCA transformation

On PCA Data

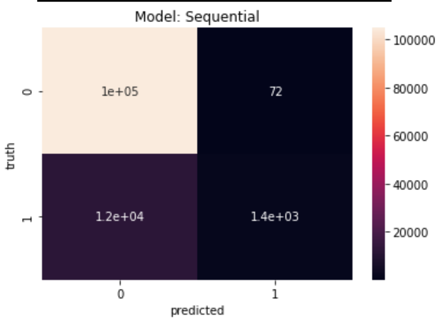
```
Accuracy: 0.9877482044782425
Sensitivity: 0.4925373134328358
Specificity: 0.9928464112545243
```



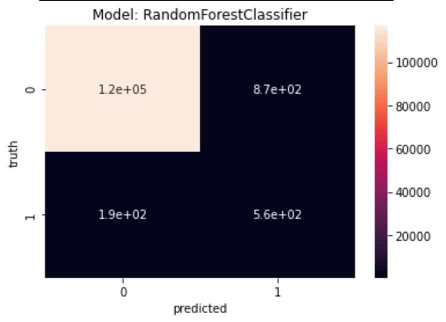
```
Accuracy: 0.9779636670891424
Sensitivity: 0.2830258302583026
Specificity: 0.9942493946731235
```



```
Accuracy: 0.8975073933248838
Sensitivity: 0.1013563869429125
Specificity: 0.999313841344871
```



```
Accuracy: 0.9910435149978877
Sensitivity: 0.75
Specificity: 0.9925684063738245
```



Section 4: Feature Engineering and Improved Modelling

Feature Engineering

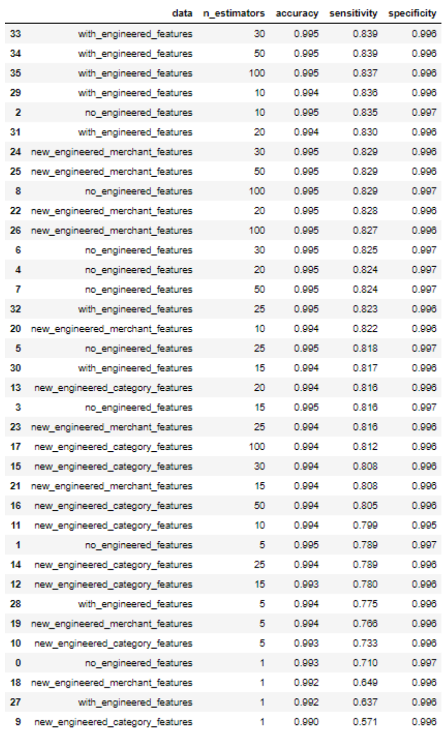
It was seen through EDA that the increase of the *amount* feature for each *merchant* and *category* differed between fraudulent and non-fraudulent transactions. It seems logical that a fraudulent transaction could be identified by not just a larger amount than a usual transaction, but a larger amount than would be expected for the specific *category* or *merchant*. For example, £300 spend on bars/restaurants might indicate a fraudulent transaction, but a £300 spend on a flight (transport) might not. The features that were to be engineered linked *category* and *merchant* to *amount*, these were the following:

- The proportion of the expected increase of a fraudulent transaction for the transaction's category
- The proportion of the expected increase of a fraudulent transaction for the transaction's merchant

These variables replaced the *category* and *merchant* features, which had the advantage of also avoiding having to use label-encoding, which is advantageous for the reasons previously discussed.

Grid Search

Grid search is a method to iterate through different options for a model and output the performance metrics of each combination. In this case, the different options were which engineered features to include, and the number of decision trees to use in the Random Forest model.



	data	n_estimators	accuracy	sensitivity	specificity
33	with_engineered_features	30	0.995	0.839	0.996
34	with_engineered_features	50	0.995	0.839	0.996
35	with_engineered_features	100	0.995	0.837	0.996
29	with_engineered_features	10	0.994	0.836	0.996
2	no_engineered_features	10	0.995	0.835	0.997
31	with_engineered_features	20	0.994	0.830	0.996
24	new_engineered_merchant_features	30	0.995	0.829	0.996
25	new_engineered_merchant_features	50	0.995	0.829	0.996
8	no_engineered_features	100	0.995	0.829	0.997
22	new_engineered_merchant_features	20	0.995	0.828	0.996
26	new_engineered_merchant_features	100	0.995	0.827	0.996
6	no_engineered_features	30	0.995	0.825	0.997
4	no_engineered_features	20	0.995	0.824	0.997
7	no_engineered_features	50	0.995	0.824	0.997
32	with_engineered_features	25	0.995	0.823	0.996
20	new_engineered_merchant_features	10	0.994	0.822	0.996
5	no_engineered_features	25	0.995	0.818	0.997
30	with_engineered_features	15	0.994	0.817	0.996
13	new_engineered_category_features	20	0.994	0.816	0.996
3	no_engineered_features	15	0.995	0.816	0.997
23	new_engineered_merchant_features	25	0.994	0.816	0.996
17	new_engineered_category_features	100	0.994	0.812	0.996
15	new_engineered_category_features	30	0.994	0.808	0.996
21	new_engineered_merchant_features	15	0.994	0.808	0.996
16	new_engineered_category_features	50	0.994	0.805	0.996
11	new_engineered_category_features	10	0.994	0.799	0.995
1	no_engineered_features	5	0.995	0.789	0.997
14	new_engineered_category_features	25	0.994	0.789	0.996
12	new_engineered_category_features	15	0.993	0.780	0.996
28	with_engineered_features	5	0.994	0.775	0.996
19	new_engineered_merchant_features	5	0.994	0.766	0.996
10	new_engineered_category_features	5	0.993	0.733	0.996
0	no_engineered_features	1	0.993	0.710	0.997
18	new_engineered_merchant_features	1	0.992	0.649	0.996
27	with_engineered_features	1	0.992	0.637	0.996
9	new_engineered_category_features	1	0.990	0.571	0.996

Figure 18: Grid Search Results

Section 5: Analysis of Final Model

Final Model:

Grid search revealed that the best version of the random forest used all engineered features with 30 decision trees. The final sensitivity was 83.9%.

Interpretation

Other than its performance, the random forest fulfils the objectives by being a very interpretable model. A random forest is just an extension of the decision tree algorithm, by creating many trees that then vote on the class to prevent overfitting. It is a very intuitive model that provides a flow-chart like process for classification. Individual trees can be extracted and followed from start to finish, and the since the rules are based on information-gain, the feature importance can also be extracted. It can be shown that the amount of the transaction and the two engineered features for merchant and category were the 3 most significant features.

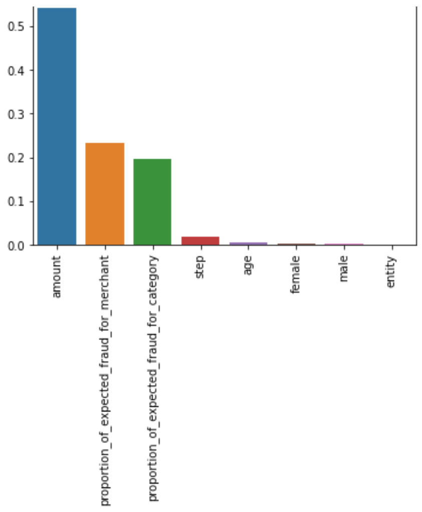


Figure 19: Feature Importance

Figure 20: One Full Decision Tree from Random Forest

Implementation into Bank Strategy

The model is by no means ready to be used as a sole mechanism with which to classify fraud. Besides the sensitivity being too low, it is also missing crucial features which are commonly used to identify fraud, such as whether a transaction is requested from a customer's country of residence, which would be a relatively simple feature to engineer. Theoretically, if this model were used, it would run alongside a live SQL database which contained the original features used in this project. When a transaction is requested, if it is classed as fraudulent, the transaction would be postponed until a user verifies their identity. The speed of the random forest is key here. KNN, for instance, would have to compute the distance between the new data point and every other datapoint, which would be very computationally expensive as the dataset grows. To improve the model, it should be trained offline, adding data over a period containing confirmed fraudulent and non-fraudulent transactions. A data science team should inspect and wrangle the data to ensure that it is suitable to be added to the model.