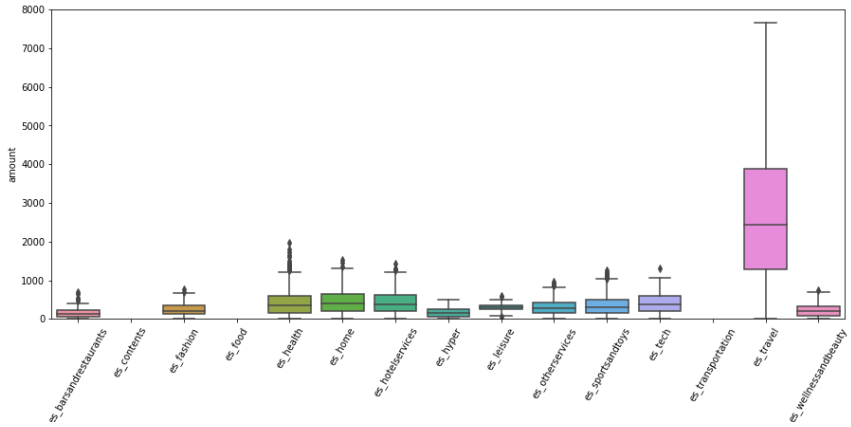
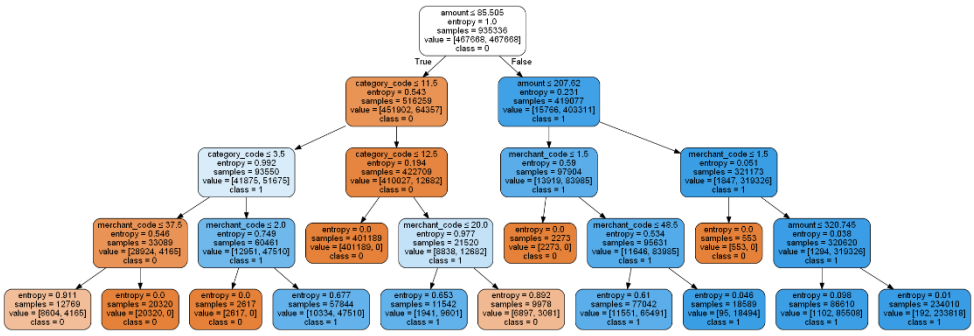
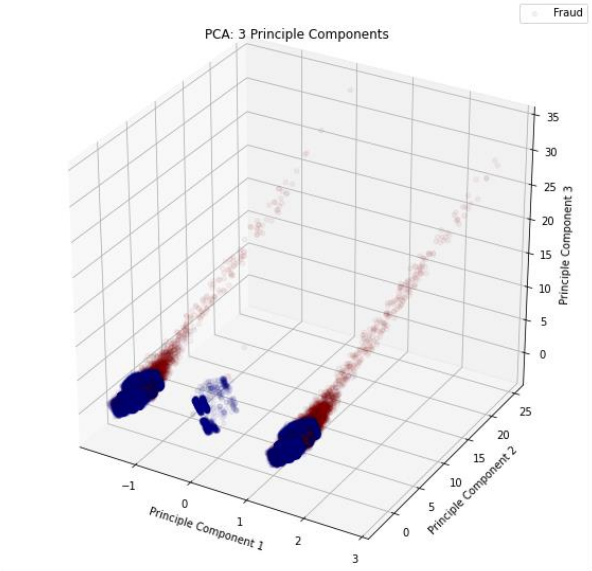






Fraud Detection

CLASSIFICATION MODELS: KEY CONSIDERATIONS AND IMPLEMENTATION

Rory Cox | 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 implies that whichever model is used should be interpretable, so that the bank understands how and why the model classifies the data. This report will detail the process and results of the project, discussing the decisions that were made to best fulfill the 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 descriptions of the variables, and while some are self-explanatory such as "amount" being the value of the transaction, others are more ambiguous or contain ambiguous values. Loading the dataset and inspecting the names and values that each variable can take leads 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, but since all values are the same as zipcodeor the variable is redundant.
- *category* – the category of good/service for which the transaction was made.
- *amount* – The value of the transaction.
- *fraud* – target variable indicating 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 their values. Addressing these issues was essential in building a viable machine learning model. Firstly, the variable *age* did not contain an integer value that would correspond 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 view 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 assumed 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 indicates 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" signified "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 to a variable, and “E” signified an enterprise. 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.

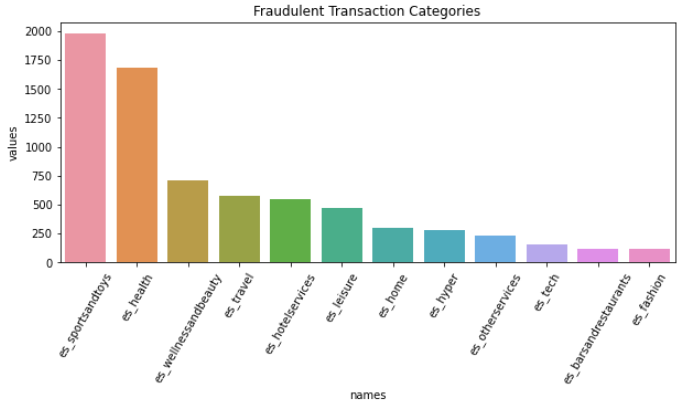
Variable Distributions

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 counts of a certain value in a continuous variable that could indicate a fault in calculations. Finally, exploring and understanding the data may reveal opportunities for feature engineering.

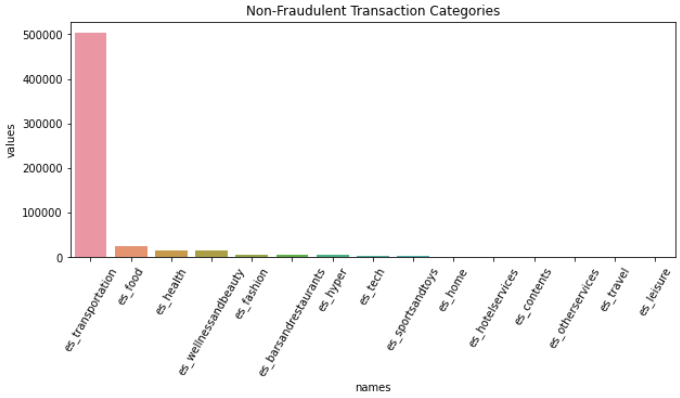
Firstly, the number of fraudulent transactions in the dataset was much lower than those which were non-fraudulent. Fraudulent transactions make up about 1.2% of the data.

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

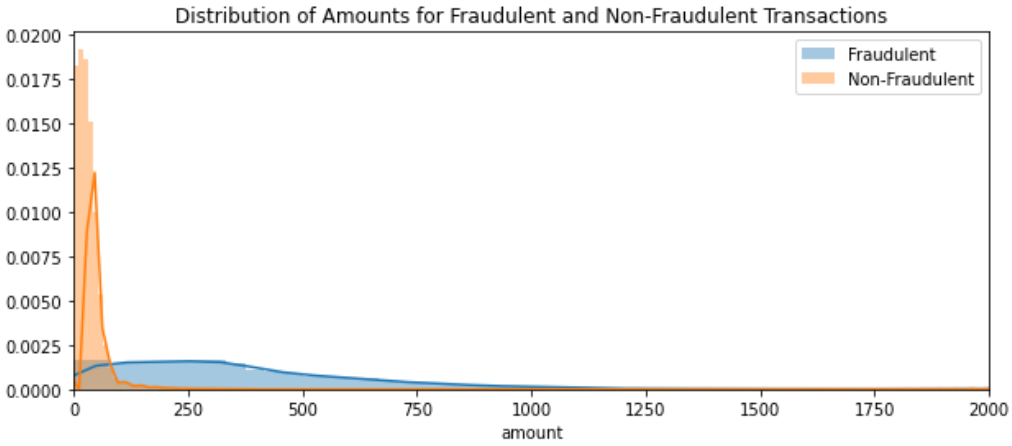
The first observation is that fraudulent transactions were spread across many categories, with sports, toys and health being particularly prominent.



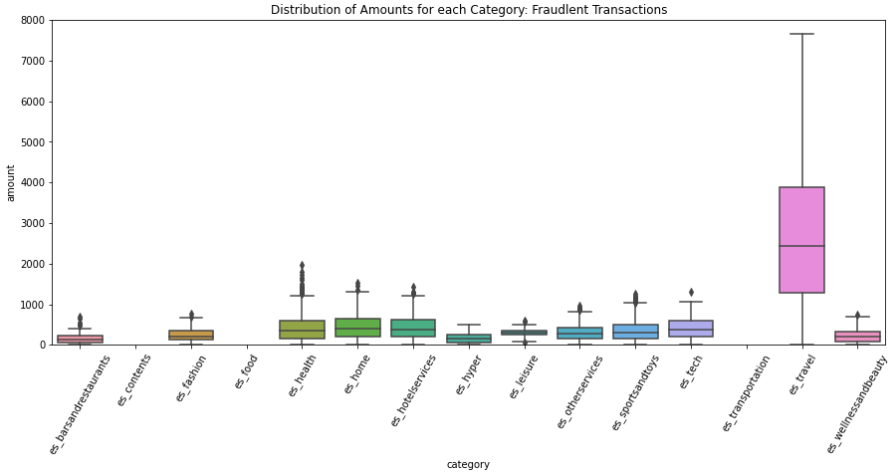
By contrast, non-fraudulent transactions were dominated by transportation.



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

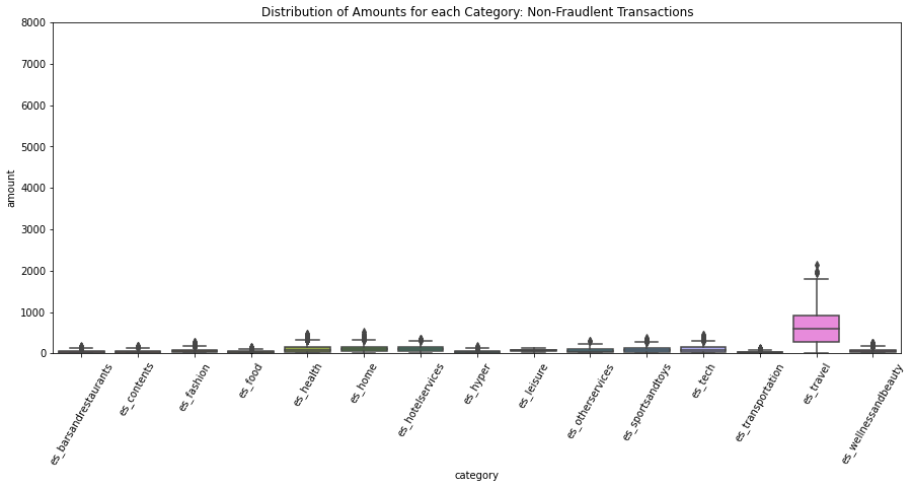


Comparing the distribution of amounts by category for fraudulent and non-fraudulent data reveals that the general increase in amount for fraudulent data is not even across categories. In particular, travel is disproportionally higher amongst fraudulent transactions, while bars and restaurants see a much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

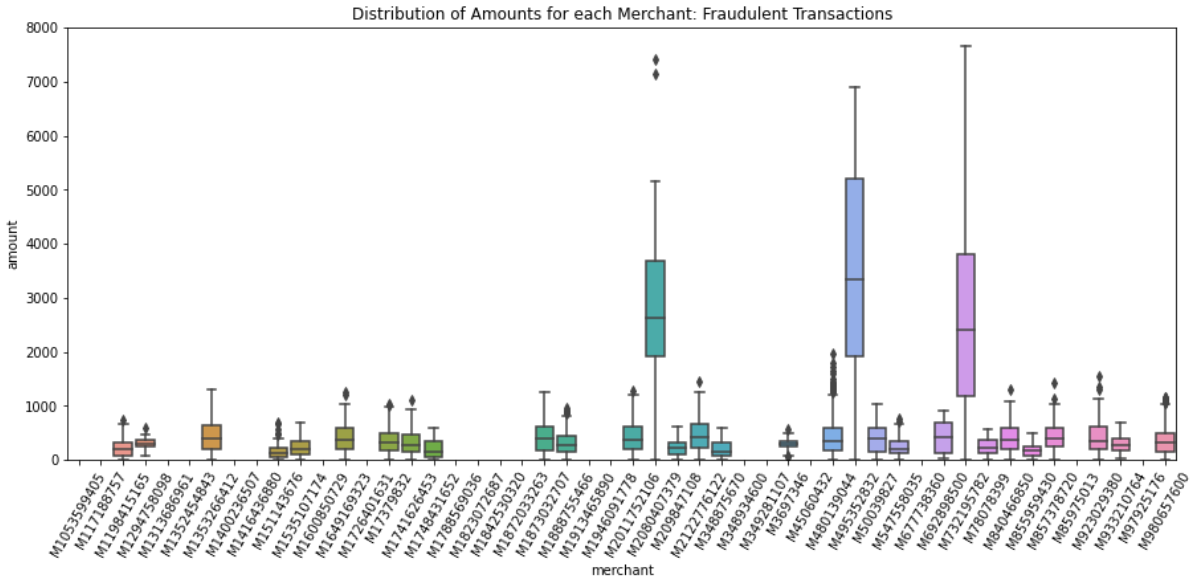


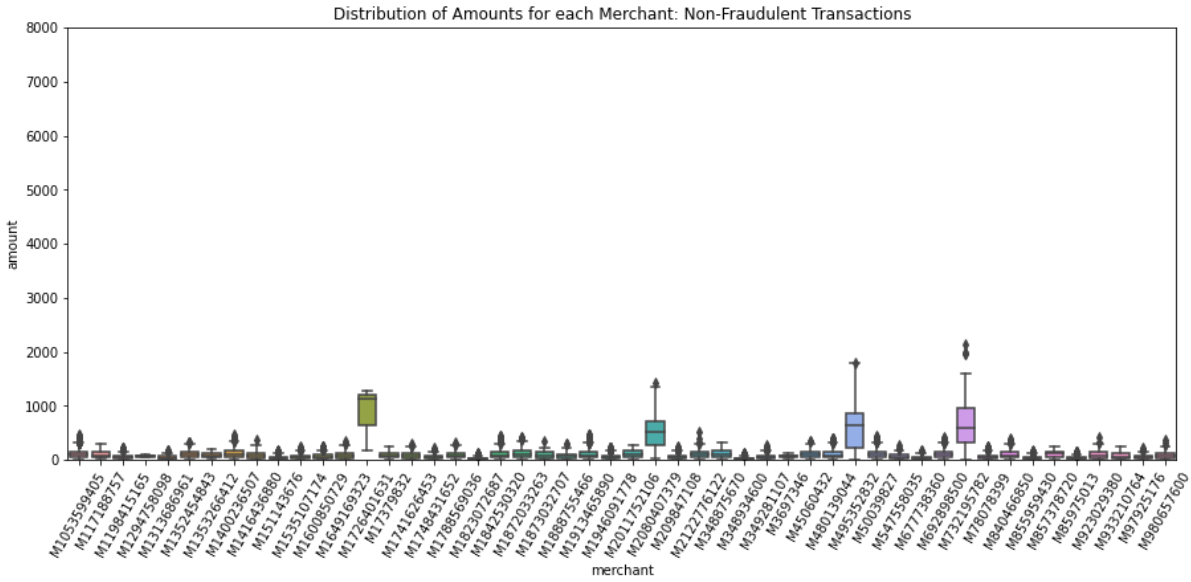
It should also be noted that some categories contain no fraudulent transactions. This is a limitation of the data, it is of course possible for fraudulent transactions on food and transportation, though since the data contains

no fraudulent data in these categories, if a model learns to never classify such categories as fraud, this detrimental aspect of the model will not be negatively impact the model's accuracy.



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





Section 2: Preprocessing

Prior to building any machine learning models, a number of decisions must 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 numeric data, and are not equipped to learn from categorical data. As such, it is important to convert categorical data to numeric prior to using a model. A number of variables in the data are 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 (whether or not the transaction was fraudulent). 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 value “1” if the category is true, and 0 if false (n – 1 categories is required, as, for one of the variables, the absence of all other variables implies the presence of the other). For the “Gender” column, which was reduced to 4 categories “M”, “F” and “E” and “U”, one-hot encoding is a viable solution, however the “category” and “merchant” features have significantly more variables, and one-hot encoding would result in a huge expansion of the feature space, resulting in the issues caused by the “curse of dimensionality”. As the number of variables grows, the feature space grows exponentially, and therefore the number of data points required to fill the feature space also grows exponentially. If insufficient data is used, models are likely to “overfit” the data, that it, fit models based on noise in the data rather than on meaningful relationships. It would also result in an enormous increase in the computational expense of training the models as the dataset grows. As such, the gender column is a one-hot encoded but the “category” and “merchant” variables are not.

Label Encoding

Another way to deal with categorical data in the “merchant” and “category” features is 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 comes with issues. Machine learning models will interpret the feature values as ordinal values on some scale, despite the label values being completely arbitrary. For example, if category “C” is labelled “3” and category “F” is labelled “6”, the values may be interpreted as category “F” is twice the value of category “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 distances based on their values which will affect the outcome of classification. When the number of labels is small, a model may still be able to derive meaningful relationships from the effects of the individual values, but as more labels are added, individual values contribute less and their relative position on the scale of values will impact models more. A superior solution is 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 overfittitng as a result of the “curse of dimensionality” discussed previously. Principle component analysis is a dimensionality reduction technique that extracts uncorrelated variables (principle components) from the 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), but this dataset has a relatively small number of features, and since features are not strongly correlated, the variance explained is low when using 2 or 3 principle components.

```
pca_1.explained_variance_ratio_
```

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

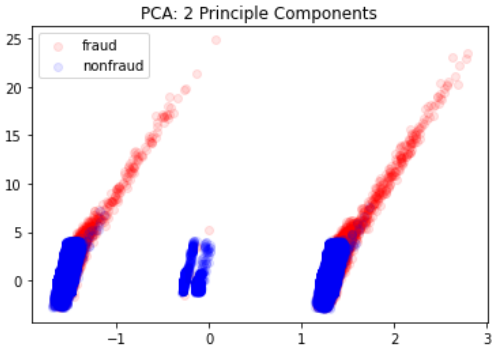
```
pca_2.explained_variance_ratio_
```

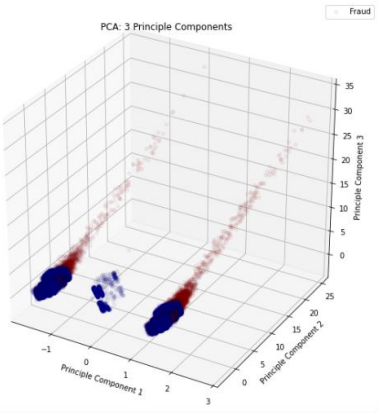
```
array([0.24952704, 0.16516261, 0.12990119])
```

To achieve 80% variance explained, more than 6 principle components is required, which defeats the purpose of dimensionality reduction on a dataframe that already contains 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; PCA would not be able to extract linear relationships for use in principle 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. However, in this example, PCA is not as effective, and while some distinction can be seen between fraudulent and non-fraudulent data, there is too much overlap to be considered distinct clusters.





However, the components themselves, being orthogonal transformations of variables composed using eigenvalue decomposition, are not intuitive to understand or compare to real world examples. Ultimately, PCA

for the reasons discussed, should not be applied to this data, though the results of the models after using PCA are available in the notebook for comparison.

Features to Exclude

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

Unbalanced Data

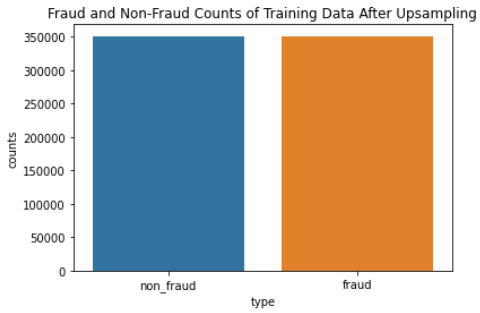
The balance of the data is an important factor to address prior to training models. The data provided shows a strong imbalance of fraudulent and non-fraudulent data, with non-fraudulent data making up 0.12% of the data. This presents two problems:

- There may be insufficient data for the model to learn the characterizes of fraudulent data
- 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 testing data by resampling with replacement. In this case, to attain an equal proportion of fraudulent and non-fraudulent transactions, the fraudulent data is resampled with replacement to match the number of non-fraudulent transactions

Oversampling will crucially only be used on the training data. The testing data must not be contaminated by oversampling since it introduces further uncertainty about our model, for example, if a certain type of fraudulent transaction is repeated many times, the accuracy may overestimate or underestimate the effectiveness of the model by measuring its performance repeatedly on the same kind of fraudulent transactions. After oversampling, the distribution of the training data is equal.



Section 3: Modelling

Approaches the Modelling

There are four criteria for a desirable model in this context of this fraud detection task: Accuracy, Sensitivity, Scalability and Flexibility. Accuracy is the most obvious desirable quality of the model, which is the rate at which a model correctly predicts the correct class. 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 is clearly more important for the model to correctly identify fraud than correctly identify non-fraud. For example, it may be frustrating or costly for the bank to perform security checks on customers whose transactions are incorrectly flagged as fraudulent, but this is 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's is designed to identify spam and remove it from an inbox, but it is more important to avoid removing genuine mail from an inbox.

The third factor, scalability, is particularly important for such a large dataset, and particularly if the bank wishes to maintain and grow the dataset over time and train models on the new data. Many machine learning models suffer from exponential increases in computational expense when training on more data.

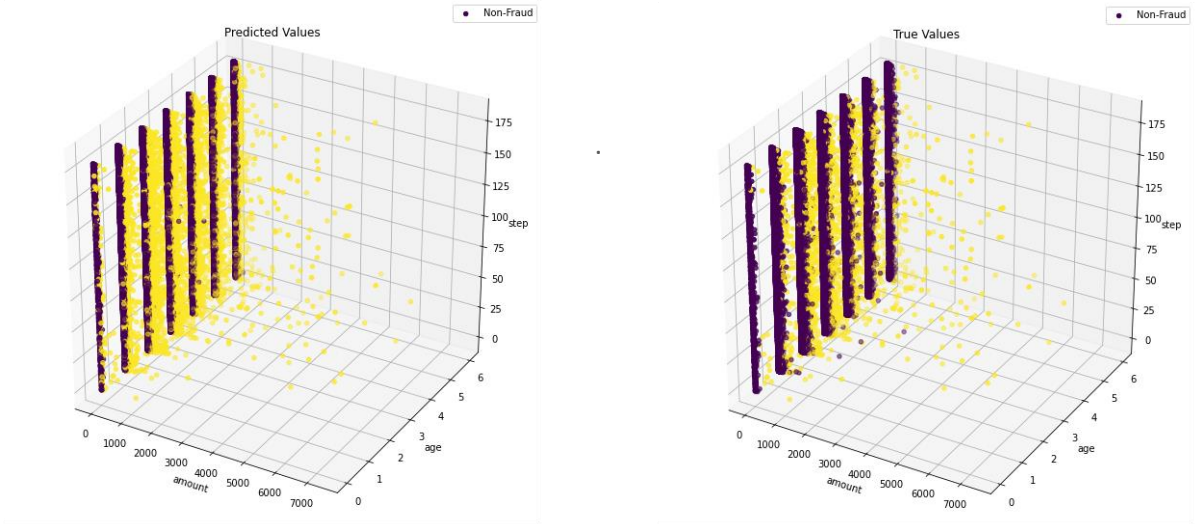
The final factor, interpretability, is a specific criterion for this task. The bank wishes to be able to understand how the model is using the features it has provided to classify transactions. This may be for their own sense-checking or some other practical purpose. The bank may wish to be able to provide 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 rejecting on the basis of ethnicity, gender or religious bases. Interpretability in both cases is key to provide such information. 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-Neighbours Classifier (KNN)

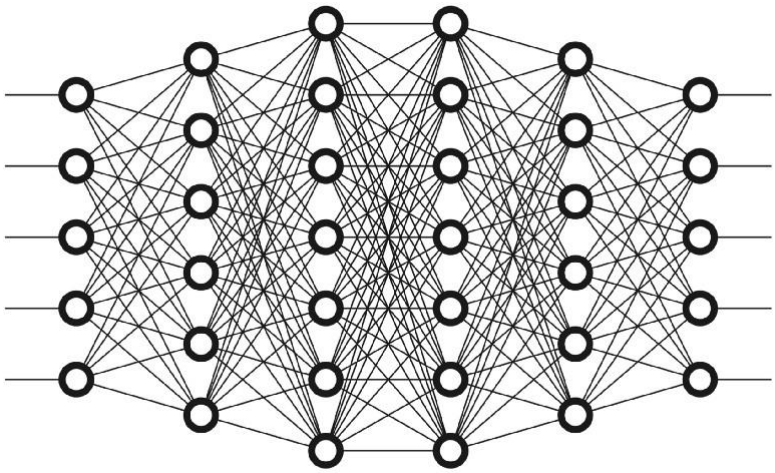
K-Nearest-Neighbours is an algorithm that classifies data based on its proximity to other data points. After scaling all the features, the algorithm computes the distance from each point to every other point. Whichever class has the largest number of the closest n points is the class assigned to the point. This model is very intuitive, using the logic that similar classes have similar values as features. Though it is not possible to visualize beyond 3 dimensions, visualizing the results of KNN in any combination of up to 3 dimensions chosen from the feature set is a simple task. The main drawback of KNN is that the ability to scale is very poor. Since each 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 location of classes.



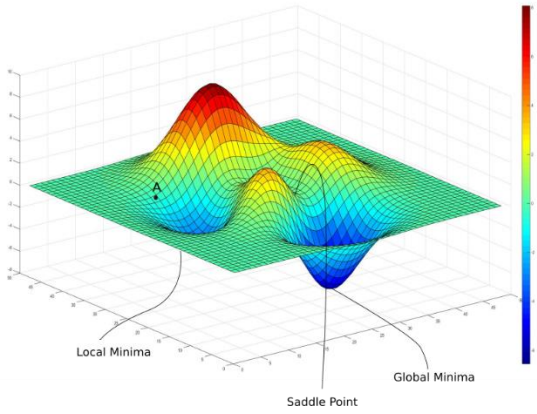
Neural Network Classifier

A feed-forward linear neural network can also be used to classify features. 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 is 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 incorrect the network was. The model then performs “backpropogation”, which is using stochastic gradient descent to adjust the weights of the layer so as to achieve a more accurate result. In Theory, any function can be modelled by a sufficient number of layers.



Neural networks are limited in this situation for being most like a “black box” model, 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 between each node and layer, except by referring to backpropogation. The number of layers

to use are a subject of academic research, 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 if the learning rate is too high.

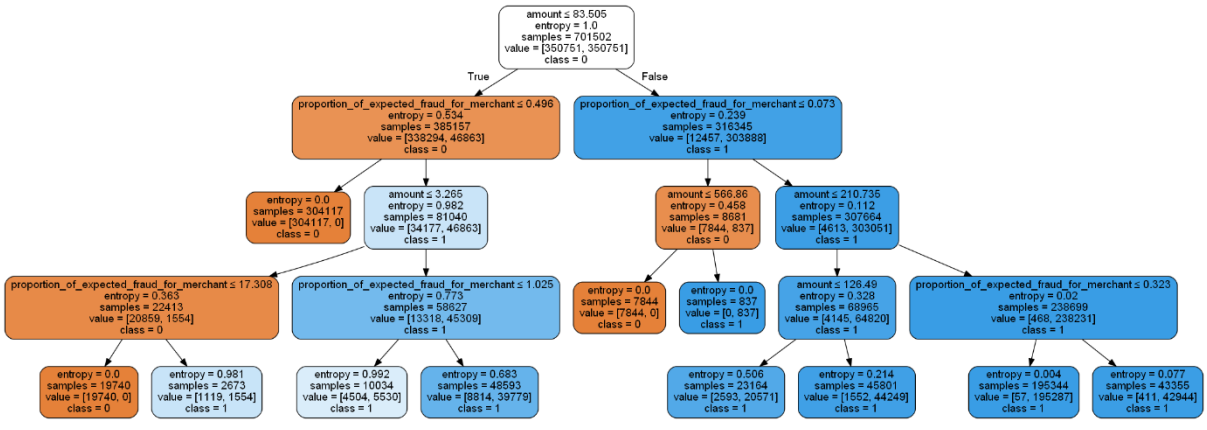


The computational expense of the neural network depends on the number and size of the layers, however it scales linearly as data increases, since backpropogation is performed after every batch of inputs.

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 results of the question “yes”, “no”. Finally, the leaves of the tree are the resulting classes. Without going into excessive detail, the decision tree chooses the rules by assessing the “information gain” for each variable at each point in the tree. Information gain is how well a given variable separates the values into the target classes in the training data. 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 adequate for visualization. A user can trace the process from node to node until they reach a leaf, and visualization makes this much easier. Decision trees also scale very well with large datasets.

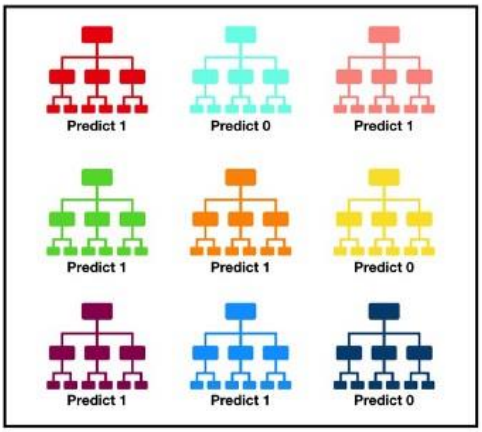


One common drawback however is the tendency of decision trees to overfit, though this issue is somewhat well addressed by instead using a Random Forest model.

Random Forest

Random forests are an extension to the decision tree algorithm. A random forest creates a large 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.



Tally: Six 1s and Three 0s
Prediction: 1

Comparison of Initial Results

Best Initial Model

The best model is the random forest

Interpretation

See feature importance
Single trees from the random forest are easy to see
The rules can be extracted

Feature Engineering

It was seen in 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 that 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 will be engineered link category and merchant to amount, these are 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 will replace the “category” and “merchant” features, which has 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 will be which engineers features to include, and various values for the number of decision trees to use in the Random Forest model.
Grid search reveals that []:

	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

Final Model:

Final best model is random forest with engineered features. The engineered features ignore the possibility of

Conclusions

Implementation into Business

The first point to make is that this model should not be used as a sole mechanism with which to classify fraud, and is also not in a good state to be ready to support existing procedures. Crucial aspects are missing from the data such as an adequate range of zip codes for the transaction, so that a feature could be engineered to identify whether the customer is using their account in their country of residence. This is a feature that could be easily engineered with a large dataset of international records of transactions. [X]% sensitivity is also not an adequate success rate for something as important as fraud classification.

Theoretically if this model were to be used, it would run alongside an a live SQL database that received data on transaction requests. If the model classified a request as fraud, then the user would be prompted to enter a passcode, answer a security question, receive a call from the bank or otherwise prove their identity before the transaction is approved. Ideally, we would want to improve the model by training it on new data, but it would be important not to train the model live. In between each iteration of training it would be important for a data science team to assess the data to ensure it is reasonable to use it for training. Also, the model should not recycle its own verified classifications. If the model adds the verified non-fraudulent and verified-fraudulent data to it's own training, without adding a proportional number of transactions that were misclassified and verified as fraudulent or non-fraudulent after a period of time, then the model will overtrain on data that it already understands how to classify effectively.