I. Data Pre-Processing

Functions used for data pre-processing can be found in Part1/preprocessing.py

A. Re-Label

Column `simple_journal` has the payment status.

- drop rows which have `simple_journal` = "Refused", since we are not sure whether it was fraud.
- replace `simple_journal` = "Chargeback" with `1`, i.e. fraud.
- replace `simple_journal` = "Settled" with `0`, i.e. not fraud.

B. Check for NaNs

Remove NaNs from `issuercountrycode`, `shoppercountrycode` and `cardverificationcodesupplied`

We replace NaN with `False` for `cardverificationcodesupplied` since we assume missing value means card verification code was not supplied.

Rest are replaced with dummy values which we numerically encode below

Remove NaN in `mail_id`

The NaNs in `mail_id` column are strings "na", which we replace with dummy value "email99999" and later encode it numerically

C. Convert `creationdate` and `bookingdate` to a pandas datetime object

This allows more flexibility later for date wise aggregation using datetime functions

D. Encode `issuecountrycode` and `shoppercountrycode`

- First we create a country code list that encapsulates both shopper and issue country codes
- Then we encode `issuecountrycode` and `shoppercountrycode` using this full list

E. Convert transaction amount in `amount` to EUR

- The transaction amount in the `amount` column is in the currency according to the `currencycode` column
- We convert all transaction amounts to euros using a fixed exchange rate

F. Replace `accountcode` with the respective Country Code

Since each account code is associated with a country. We extract this and encode it later.

G. Replace all `cvcresponsecode` values above 2 with a single value

`cvcresponsecode` 3 - 6 all correspond to not checked and can be merged into one

H. Numerical Encoding of categorical columns

We can replace the categorical columns with positive integers, since we need numbers to input to the ML Model

---

II. Visualization

A. Distribution of the Transaction Amount

- We analyse the distribution of `amount_eur` column for fraud and non-fraud cases
- We hope to find a significant difference between them which will allow us to classify the cases well
- To do this, we make a boxplot to see the quartiles and outliers

We can see here that there are outlier values for the non-fraud case. For further analysis, we remove the records where an amount > 180 is spent in a non-fraudulent transaction.

Now, if we compare this to the distribution of transaction amount in the fraudulent cases, there is a significant difference in distribution

---

B. Heatmap

TODO: Put explanation here

C. Absolute Difference in Correlation

TODO: Put Explanation here

D. Visualize currencycode

Are transactions made in some currencies pre-disposed to having more fraudulent transactions?

We can see that transactions with `currencycode` = 1 have higher likelihood of being fraudulent.

E. Visualize Card Type

Are transactions made using some credit card types pre-disposed to having more fraudulent transactions?

`txvariantcode` = 2 and 8 have a higher likelihood of being fraudulent

F. Visualize Card Issuer

Do some transaction made using cards from certain card issuers entail more fraud?

We can see that some card issuers do have a tendency to accrue higher fraudulent transactions from their cards

---

G. CVCResponseCode

Are transactions that have certain cvcresponse codes pre-disposed to being fraudulent?

We can see that the cvcresponsecode 0 is more prominent in cases of fraud, whereas 1 is more prominent in non-fraud cases

---

H. Isomap

TODO: Put short description of why we are doing this

TODO: add explanation of the result

III. Subsampling the Majority Class

The class imbalance between fraud and non-fraud is quite stark. We see the distribution below.

We would like to retain all fraud samples while sub-sampling about 10% of the non-fraud cases. This should be enough to train the models without encountering problems of bias.

We do this below:

III. Rank Swapping

Script used for rank swapping can be found in Part1/rank_swapping.py

TODO: add more details here

V. Feature Engineering and Standardization

Here, we create features by transforming and/or combining existing features.

A. countries_equal: Shopped Country = Country in which Card was Issued

The intuition is that if the country in which the money was spent is not the same as the country in which the card was issued, it is likely that the transaction was fraudulent.

B. Time of Transaction: Day of Week and Hour of Day

Visualize if fraudulent transactions happen at specific times or hours of the day

There is a clear difference in distribution between fraud and non-fraud. Fraudulent transactions tend to be distributed throughout the day, whereas non-fraudulent transactions gradually increase as the hour passes.

C. Aggregate Features

Script used for creating aggregate features can be found in Part1/aggregate_features.py

We engineer the following features by aggregating over time for credit card. We use transaction aggregation strategy as a way to capture consumer spending behavior in the recent past. We put each credit card transaction into the historical context of past shopping behavior. In essence these attributes provide information on card holders' buying behavior in the immediate past.

- `prev_month_avg_amount` : Average amount spent in the previous month
- `prev_week_avg_amount` : Average amount spent in the previous week
- `prev_day_amount` : Average amount spent in the previous day
- `daily_avg_over_month` : Average amount spent per day in the last month of transactions
- `prev_day_same_country` : Number of transactions in the last 1 day in the same country as in this transaction
- `prev_month_same_country` : Number of transactions in the last 1 month in the same country as in this transaction
- `prev_month_avg_amount_same_country` : Average amount of transactions in the last month in the same country as in this transactions
- `prev_month_same_currency` : Number of transactions in the last 1 month with the same currency as in this transaction
- `prev_month_avg_amount_same_currency` : Average amount of transactions in the last month with the same currency as in this transactions
- `prev_total_transactions` : Total previous transactions with this credit card

NOTE: the cell below will take about 6 minutes

We visualize some of these below to show that they have different replationship with the label. These features are much lower and distributed more tightly around 0 for fraudulent transactions as compared to non-fraud transactions.

Feature Normalization

We normalize the features by making the data zero mean and unit standard deviation. This is because most ML algorithms assume that the data is distributed normally and work well when data is distributed normally around 0.

K Fold Cross Validation

We create our 10 fold cross validation set here. We use this for training and validating our models below

VI. SMOTE

Script used for SMOTE can be found in Part1/rank_swapping.py

SMOTE is a technique to synthetically generate samples of the minority class so that they are similar to the existing samples in a vector space of their attributes

Below, we show our experiments. We train 3 models: Decision Tree Classifier, Naive Bayes Classifier and a Support Vector Classifier, on the SMOTEd and unSMOTEd data and compare the True/False Positives and corresponding AUCs.

For the unSMOTEd data, the class imbalance is 99.44% to 1.56%.. For the SMOTEd data, we restore the class balance to 50-50.

We were not able to train the SVC classifier on the SMOTEd data because the upsampling increase the dataset size too much that training takes forever.

In our github repository: - Part1/models.py contains our models - Part1/metrics.py computes our metrics - Part1/visualization.py plots the AUC curve and feature importances

A. Decision Tree Classifier

unSMOTEd data

SMOTEd data

B. NaiveBayes Classifier

unSMOTEd data

SMOTEd data

C. Support Vector Classifier

unSMOTEd data

SMOTEd data

D. Results

According to the results above, the support vector classifier performs the best among the three. It has an average AUC of around 90. It finds 101 positives accurately with only 2 false positives.

The NaiveBayes classifier is quite good as well since it gives an AUC of 86.

SMOTE is generally a good idea since it: - allows us to increase the number of minority samples - does not repeat the same samples while over sampling - smartly oversample by interpolating between nearest neighbours to maintain the data distribution

SMOTE does not work in all cases, namely for simple Decision Trees since they are a bit different from other nonparametric statistical methods in that they cannot generalize to variations not seen in the training set. This is because a decision tree creates a partition of the input space and needs at least one example in each of the regions associated with a leaf to make a sensible prediction in that region. Since we do not do SMOTE for the validation/test set it performs poorly.

One should use forests or even deeper architectures instead of trees, which provide a form of distributed representation and can generalize to variations not encountered in the training data