Q4 Report

This question intends to predict fraudulent credit card transactions. The dataset contains only 0.75% of fraudulent transactions, while the entire dataset has over 1.2 million training instances. The following sections will explain my approach in feature engineering and model hyper parameter tuning. Going into this question my approach was to try and build a rule based model, hence building good features was important.

The following libraries were used from sklearn:

Decision Tree Classifier, KMeans Clustering, KFold validation, GridSearchCV

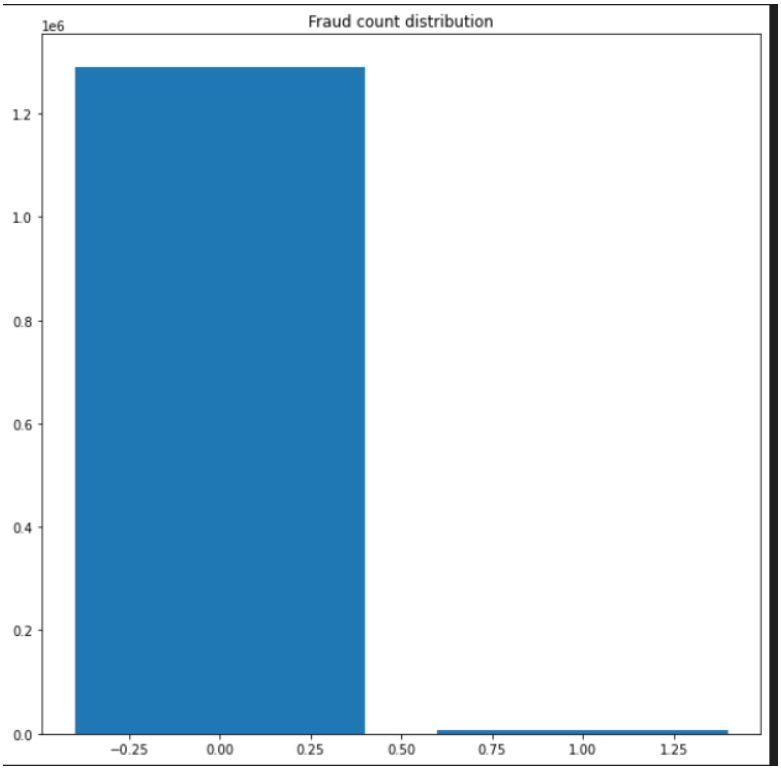
The following statistical libraries were also used from scipy:

Chi2_contingency, chi2, f_oneway

As 'trans_num', 'first' and 'last' do not have any useful information, they were dropped off.

Distribution of the classes in the dataset was drawn as presented in Figure 1

Figure 1



In order to avoid any Data Leakage, the dataset is split into a training set and testing set. All Exploratory Data Analysis was done using the training set. The split was done using a stratified k fold split to ensure an equal percentage of the 0 and 1 classes.

Merchant and Street attributes were not considered valuable hence were dropped off the dataset.

Transaction time was very interesting to me. Hence after transforming it to datetime format, I made 5 different categories of 'Off Hours' from 8 pm to 8 am, 'Morning' from 8 am to 12pm, 'Noon' from 12pm to 4pm and 'Evening' from 4pm to 8pm.

As shown in the following contingency table 5358 credit card frauds occur in the Off hours. This is 89% of all frauds in the training dataset.

TimeofDay is_fraud	Evening	Morning	Noon	Off Hours
0	209828	135260	208271	477976
1	250	145	252	5358

A function was written to find the chi2_correlation between the categories and the number of frauds that occur (refer Jupyter notebook for function). As per the chi2 test, the critical value for a Degree of Freedom of 3 (as per above table) and 0.05 significance is 7.81. However the chi2 statistic for the above counts is 4411. Hence null hypothesis is rejected and we conclude that the **new engineered attribute** is statistically important.

Typically credit card frauds tend to have transaction amounts which deviate from the normal transaction amounts. In this dataset, as we have different categories of transactions, the Median transaction amounts will differ too. I have used median instead of mean, as median is not affected by outliers(in this case fraud transaction amounts) like the mean value is.

I have grouped by on category and gotten the 25^{th} , median and 75^{th} percentiles for each category. Then an inner join was done on "category" with the training dataset and category dataframe.

Then if a transaction was above 75% + 1.5IQR and less than 25% -1.5IQR, I flagged these as "Outside upper/lower bounds". This was also an **engineered attribute**

Afterwards I check every record which has a Yes on "Outside upper/lower bounds" and Yes on "Off Hours**". It was observed that this subset of instances consist of 4537 instances of Fraudulent transactions out of the total 6005 instances in the training dataset. This is 75.5% of the Fraudulent cases, showing these 2 engineered features are extremely important.**

Then I have looked at on a state level basis. It was observed that states like NY, PA, TX have higher counts of credit card fraud. Hence to engineer another attribute, I have grouped on state and found the total fraud count for each state.

As there are 51 states, I did not want 51 different one-hot encoded attributes. So I wanted to categorize these. I applied KMeans clustering on the states total fraud counts to get 4 different clusters. It was clearly seen that states like NY,PA,TX form there own cluster, while states with low fraud counts create there own cluster (Refer jupyter notebook for code and result). I mapped the 4 clusters into 4 risk levels 'Risk 0', 'Risk 1', 'Risk 2', 'Risk 3'. This too was merged with the training data using an inner join.

Given that I have gotten the information from State, I dropped off state, zip, trans_date_trans_time and dob.

Then I was curious to check if there was any useful information in the distance of a transaction from the users registered location. For example, do fraud transactions tend to have a higher mean distance from registered location compared to normal transactions.

Using the Haversine equation [1], the distance between latitudes and longitudes of the merchant and registered location was calculated. Then I used the Anova Oneway test to check if there is any statistical difference between the distances for fraud and normal transactions. It gave a p value of 0.93 which is greater than 0.05 and not significant. Hence I didn't proceed with this attribute and dropped it off.

Finally I have created a fraudDetection class to encapsulate all the transformations that are carried out on the dataset. This class has the following functions exclusive of the constructor:

- Get_train_test_set (splits the raw dataframe which is read from the txt file into a training and test set)
- Get_grouped_category_details (used to group on category and find the 25^{th} , median and 75^{th} percentiles) (will be run only during training and preparation of training set) (For test set, this is stored as a class attribute from training stage)
- Get_grouped_state_details (used to group the data on state level and carry out KMeans) (will be run only during training and preparation of training set) (For test set, this is stored as a class attribute from training stage)
- One_hot_encode_categories (used to one hot encode Category, Gender, Time of Date and State Risk Category and drop city and job)
- Fitandtransform is the entire data processing pipeline and it returns the dataset and the labels separately.

Classifier Training and hyperparameter tuning

As mentioned earlier, I wanted to make my model rule based, hence I have first used a decision tree from sklearn library. The hyper parameter that is being tuned is max_depth of decision tree and the criterion. Gridsearch CV is used to do hyper parameter tuning with a stratified cross validation of 5 folds. As the question mentions the model will be judged based on ROC AUC, the scoring function of gridsearchCV is made to be "roc_auc". The model is trained and the best estimator is obtained from gridsearchCV. The following piece of code predicts on the training dataset using the best estimator and finds the decision probability. This is applied to the roc_auc_score from sklearn to find the training auc score.

```
y_pred_proba =
gridsearch_dtree.best_estimator_.predict_proba(processed_training_data)
roc_auc_score(training_labels,y_pred_proba[:,1])
```

The auc score is calculated as 0.997. It is a near perfect estimator as a perfect classifier would have AUC of 1.

Then I predict on the test set.
Note: The test set has had no exposure to Exploratory Data Analysis or training. Hence it is fair to assume that this is an unseen dataset.
An AUC score of 0.983 is obtained. These are excellent AUC scores using a simple decision tree, however because during feature engineering, I have engineered useful attributes, the AUC scores are very good.

As the AUC score is mostly concerned with the True Positive Rate versus the False Positive Rate, my model does not produce many false positives, causing a low FPR. Hence this allows the AUC to be high