Assignment 1 Report

Aryan Nath

October 7, 2024

Online Payment Fraud

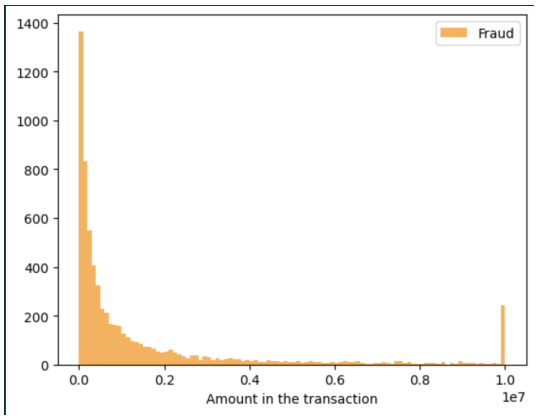
Introduction: Online payment fraud detection using the account ID of the transaction sender and receiver, the amount of transaction, the change in bank account balance, and the type of transaction.

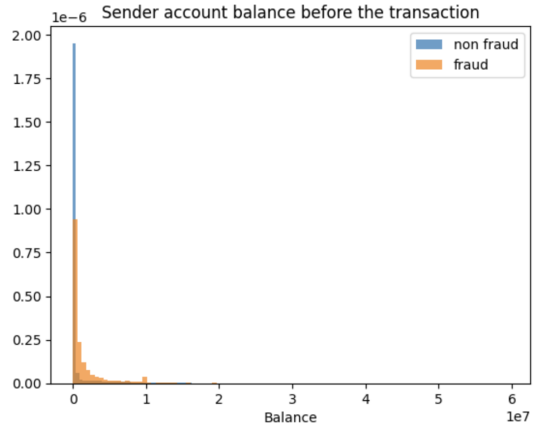
Methodology: Classification of fraud transactions based on 'amount', 'oldbalanceOrg' and two new features 'diffOrig' and 'diffDest' using decision trees. *get_entropy* : to calculate the entropy of a node. *split_data* : to split the passed data based on a feature index and threshold. $find\_best\_split$: to find the best split based on information gain. *build_tree* : used to build the decision tree with split_data and information gain. fit : fit the decision tree to the data. *pred_node* : predict the class of a node based on its majority datapoint labels. *predict* : predict the classes of datapoints in a dataset. I have filtered the dataset for NaN values and then downsampled the training dataset with respect to the minority fraud transaction datapoints to better fit on them.

Experimentation: On exploring the dataset, I found out the this: Within `type`, there are only two categories that have fraudulent cases. For fraud transactions, the new - old account balance for the sender is very low or negative at a high frequency, the same for the receiver is a low positive value at a high frequency. In non-fraud transactions, for the sender, the difference is centered at 0 at a high frequency, and for the receiver, the difference is centered at 0 with lesser deviations than the sender. I noticed that the sender's initial bank balance is more likely to be high as compared to non-fraud transactions.

The feature correlation heatmap below indicates that none of the features were significantly correlated enough to be dropped direrctly.

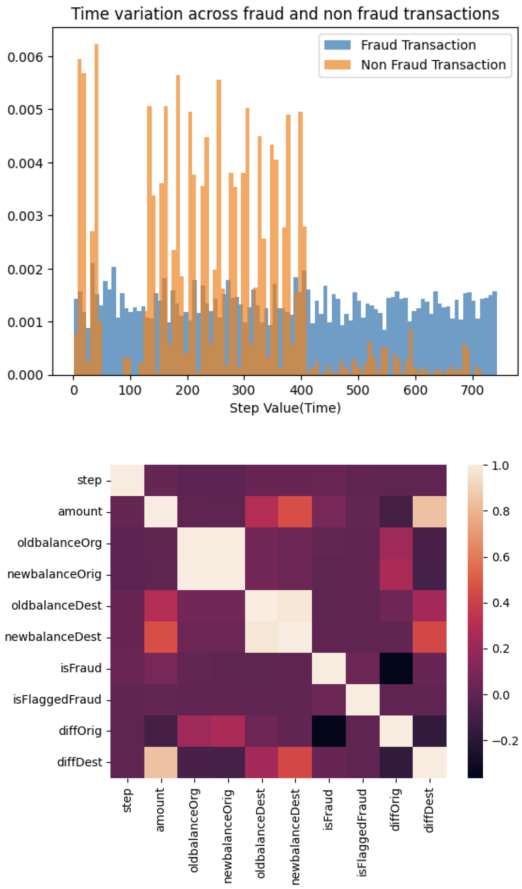
For my first model, I used `step`, `amount`, `oldbalanceOrg`, `newbalanceOrig`, `oldbalanceDest`, `newbalanceDest` and created features `diffOrig` and `diffDest` for the difference in the bank accounts before and after the transaction. The hyperparameters used were `max_depth = 20`, `min_size = 100`. The purpose of this was just to get an understanding of whether the model overfits to the data based on the max depth and how close are the datapoints based on their





number at the leaf node. The proportion of fraud transactions as compared to non-fraud transactions in the dataset is very low, and I want my recall to be very high even if there's some compromise on the precision (non-fraud transactions classified as fraud can be investigated further to be validated without necessarily causing significant inconvenience to the senders and receivers). So before training, I have downsampled my data to make the non-fraud datapoints proportional to the fraud datapoints in number.
Since the amount is not exactly corresponding to the bank account difference, the new features are not highly correlated with `amount`, as shown in the heatmap plot.

For the second model, I used the same set of features but reduced the hyperparameters to `max_depth = 15`, `min_size = 50`. I used this to sort of balance overfitting while also increasing the importance given to the label 1 datapoints for classification at a leaf node. I'm getting a recall of 0.976 on the validation dataset but a bad precision.



Based on the heatmap, there was not significant correlation between any of the features. So I considered which features to drop based on their variations indicated through histograms.

I noticed that `step` does not significantly vary for the fraud transactions as compared to the non-fraud transactions. `oldbalanceOrg` is higher in fraud transactions than in non-fraud transactions. There was not much difference in the receiver account balance after transactions in fraud transactions as compared to non-fraud transactions.

The feature selection helped me bring down the required features to `amount`, `diffOrig`, `diffDest`, `oldbalanceOrg`.

For the final model, I used the features subset `amount`, `diffOrig`, `diffDest`, `oldbalanceOrg` and the hyperparameters as `max_depth = 20` and `min_size = 30`. I'm getting a recall of 0.975 and above on a 1M size validation dataset, but

even on tweaking the parameters I can't improve the precision which was as low as 0.03. So I tried changing the downsampling, to maintain the disproportion of the two classes to an extent. I used the fraud and non-fraud proportion as 1:10 in the training dataset and now got recall 0.96 and precision 0.184. I'm giving more importance to recall than precision, so not reducing the recall further to improve precision.

Results:
The performance of the first model is as follows:
Classification Metrics:
Accuracy: 0.9794805441783416
Precision: 0.058946552361377795
Recall: 0.9918094949188533
F1 Score: 0.1112793982505701
Confusion Matrix: [[4979111, 104392] [54, 6539]]

This model was meant for a general evaluation, without any parameter optimisation. As can be seen the recall is high, facilitated by the training after downsampling but the precision is low.

The performance of the second iteration of my model is as follows:
Classification Metrics:
Accuracy: 0.982201514470454
Precision: 0.06720455037249992
Recall: 0.9892310025784924
F1 Score: 0.12585874179853337
Confusion Matrix: [[4992978, 90525] [71, 6522]]
This model accounts more for precision by increasing the number of datapoints in a leaf node for making a majority decision, that is there will be
The performance of my final model is as follows:
Classification Metrics:
Accuracy: 0.9807634276445867
Precision: 0.0627507684646985
Recall: 0.9939329591991506
F1 Score: 0.11804867503737998
Confusion Matrix: [[4985627, 97876] [40, 6553]]
Challenges:
The main problem faced with this dataset was the imbalance of the fraud and non fraud datapoints. I overcame this by downsampling the dataset to fit better to the fraud datapoints. There was also a lot of mismatch between the transaction amount and the difference in the balance of the bank accounts. While I couldn't do outlier detection on this, I created a separate feature to account for the difference in the balances of the accounts and used that along with the 'amount' feature.

Conclusion:

To summarize, the features representing the transaction amount, balance of the source before the transaction and difference in the balance of the sender's and receiver's accounts are able to get a very good recall on the entire dataset. This model can be improved by accounting for increase in precision.

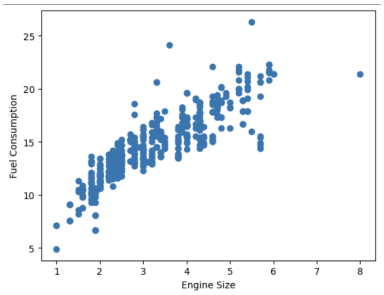
References: https://www.youtube.com/watch?v=ZVR2Way4nwQ
https://airbyte.com/data-engineering-resources/data-preprocessing
https://datascience.stackexchange.com/questions/61858/oversampling-undersampling-only-train-set-only-or-both-train-and-validation-set

Fuel Consumption Dataset

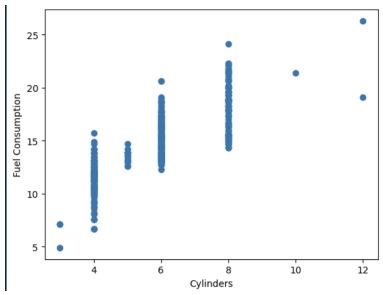
Introduction: Fuel consumption regression problem using feature dataset of various vehicles.

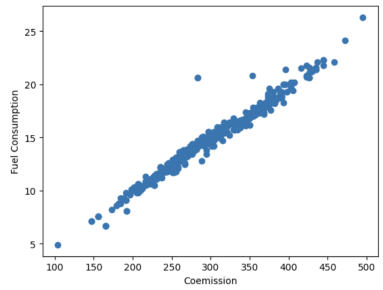
Methodology: I have implemented multiple linear regression to solve this problem using the features 'engine size' and 'coemissions'. Before feeding the data to the linear regression model, I have filtered the dataset for NaN values and standardized the feature values to account for variation in the range of the features.

Experimentation: On exploring the dataset, I found that engine size and fuel consumption shared a linear relationship. Coemissions and fuel consumption we're also highly related linearly. Cylinders and fuel consumption did not have a meaningful relationship; there are multiple fuel consumption values for the same cylinder value(indicated in the dataexploration file).



I trained my first iteration of the model on just the engine size feature with $learning\_rate = 0.000001$ and $num\_iterations = 4000$. The second iteration of my model used both features engine size and coemissions, with the $learning_rate = 0.000001$ and $num\_iterations = 6000$. The third iteration of my model used these two features, $learning\_rate = 0.000001$ and $num\_iterations = 7500$. The final iteration of my model used these two features, $learning\_rate = 0.000001$ and $num\_iterations = 6200$. Using standardization instead of normal-





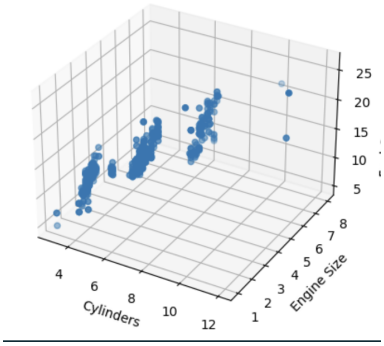
ization gave me an immense improvement in the performance.

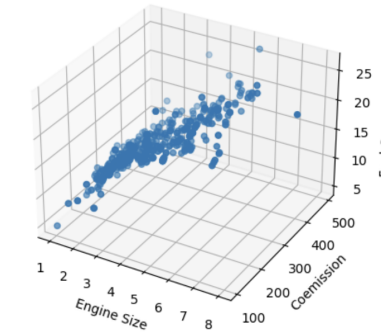
Results: For first iteration I got the following results on the validation data:
Regression Metrics:
Mean Squared Error (MSE): 0.8496076380981342
Root Mean Squared Error (RMSE): 0.9217416330502459
R-squared (R2) Score: 0.9149690272914773

For the second iteration I got the following results on the validation data:
Regression Metrics:
Mean Squared Error (MSE): 0.4577354894646821
Root Mean Squared Error (RMSE): 0.6765615193496317
R-squared (R2) Score: 0.9541886252346781

For the third iteration of the model I got the following results on the validation data:
Regression Metrics:
Mean Squared Error (MSE): 0.4331114474895654
Root Mean Squared Error (RMSE): 0.6581120326278539
R-squared (R2) Score: 0.9566530642854459

For the final iteration of my model I got the following results on the validation data:
Regression Metrics:
Mean Squared Error (MSE): 0.4711905652530642





Root Mean Squared Error (RMSE): 0.6864332198058776
R-squared (R2) Score: 0.952842005770764

Challenges:
The main challenge in the regression problem was figuring out which feature was to use. I overcame this by making scatter plots of the features against one another and seeing how proportional are their values. Another problem was finding a good learning rate and number of iterations. I did this by checking the mse and $R\hat{2}$ values for the performance of my model on both the training and validation data to check how well it was fitting to the data and whether it was overfitting on the training data.

Conclusion:
A good selection of the learning rate and number of iterations helped me get a good performing model. While I was able to get good performance on the training dataset, the performance can still be improved by looking into new features or combining features.

References:
https://www.youtube.com/watch?v=sDv4f4s2SB8
Class slides