Credit Card Fraud Detection using ML Report

Submitted to: Professor Richa Singh, HOD Department of CSE, IIT Jodhpur

---

Project By: JESIKA RAI

Introduction

In this project, we have explored different machine learning approaches for successfully identifying fake credit card details and by extension, credit card frauds. We obtained varying results from all the models. The results are summarized in the following report along with the data analysis that we performed.

Data Analysis and Visualization

After downloading the data, we checked if there were any null values in the data. There were no null values in the data. Moving forward, we plotted the correlation matrix of the data to get an idea about what features are not relevant for the classification purpose. Following is the correlation plot that we obtained, which led us to conclude that almost all the features are independent of each other and thus we decided to retain all the features of the dataset.

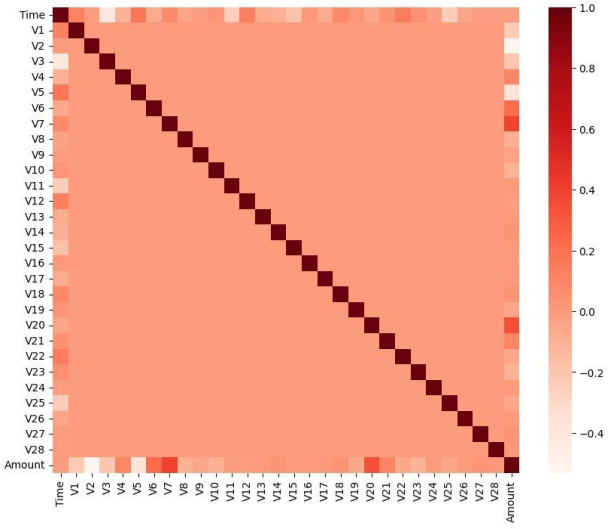

Fig. 1: Correlation Heatmap
The darker regions represent high correlation while the lighter ones represent lesser correlation. As can be seen in the adjacent heatmap, the majority of the region is light, thus indicating lesser correlation between features.

The correlation heatmap clearly shows very little correlation in the features V1 to V28. There is a slight correlation between these features and the time or amount.

To get an idea about the distribution of the features, we made feature plots but could not infer anything insightful. We observed that some of the feature plots had a roughly normal curve but the majority did not so we could not be certain that Gaussian Bayes would yield good results.

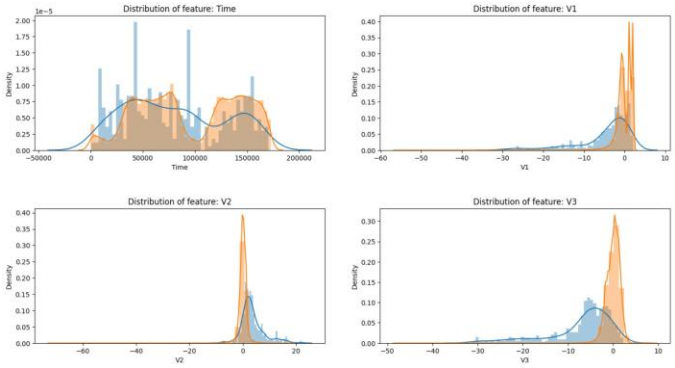

Fig. 2: Feature Distribution Plots *The plots for all the features are available in the .ipynb file*

The description of the dataset clearly mentions that the features V1 to V28 were generated from PCA so we did not go ahead with PCA. Thus, we decided to try LDA but since there were only 2 classes, LDA gave us a 1-dimensional data and on plotting that, there was a complete overlap of the two classes. Considering that the original dataset contained 30 features, scaling that down to just 1 feature obviously led to a humongous information loss, and will give bad results.

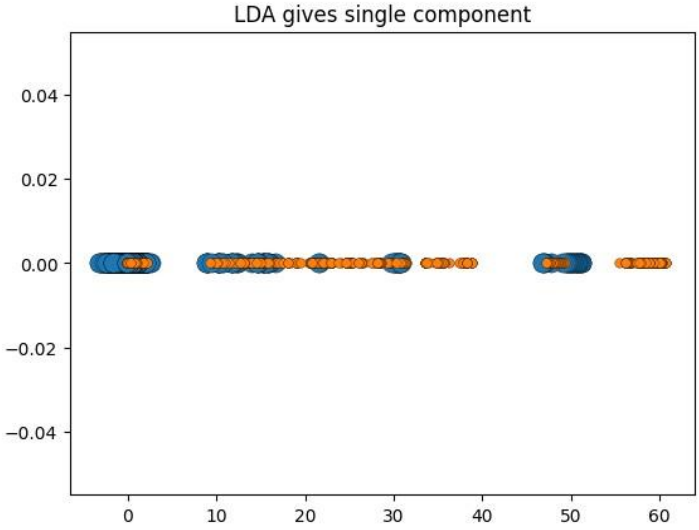

Fig. 3: LDA plot

The blue dots show the Non Fraud Transactions (class = 0).

The orange dots show the Fraud Transactions (class = 1)

Models

After studying the obtained plots and results from Data Analysis and Visualization, we decided to implement the following Machine Learning models. We implemented and evaluated all the models through the *pipeline* class (in the .ipynb file).

- Decision Tree
- Random Forest Classifier
- Bagging Classifier
- XGBoost Classifier
- Logistic Regression

- Gaussian Naive Bayes
- K-Nearest Neighbour

We observed that **tree based models** performed better than the other models. This is because the tree based models could capture the relationship between different features in a better manner as compared to the other models. This is so because all the other models are dependent in some way on the kind of distribution of the data. The **Gaussian Naive Bayes** model assumes independence and normality of the data and as already established during data analysis, the features are not normally distributed. Likewise, **K-Nearest Neighbour** simply operates based on the proximity of the points belonging to one class and as we have already observed during data analysis, many of the data points of different classes are overlapping, thus leading to great disadvantage for KNN. Since tree based models have no such assumption or dependence on the arrangement of the data points, they ended up having a much better performance.

Evaluation Metrics

The usual evaluation metric is the accuracy score but since in the case of this dataset, the *fraudulent* class had significantly less number of data points than the *non-fraudulent* one, the data was highly unbalanced and thus a minor misclassification in case of *fraudulent* class would still have given us a very high accuracy score (as high as 99.8%). Thus, we decided to drop *accuracy* as an evaluation metric.

On the other hand, a measure of true positives and negatives is a much better metric as it gives us a clear insight about the correctly and the incorrectly classified data points and by extension, an insight about the performance of the model. **Precision, Recall, F1 Score, Area Under Precision Recall Curve**, are all evaluation metrics that take into account the true positives and negatives. Thus, we decided to use all these as our evaluation metrics with Area Under Precision Recall Curve being the primary evaluation metric.

To get a better understanding of the voting classifier (for the best 3 models out of all 7) on actual and oversampled data, we also plotted the confusion matrices. We observed that on oversampling the data, the number of true positives in the test dataset increased, thus indicating that oversampling helped the classifier to learn better and provide better results.

Hyperparameter Tuning

To enhance the performance of all the models, we decided to vary the hyperparameters to obtain the optimal value of the hyperparameters so as to get the best model.

For tree based models, except xgboost, we varied the *criteria, max_depth,* and *min_samples_split.* We obtained the following optimal values for all the tree based models. DTC:

```
{'criterion': 'gini', 'md': 5, 'mf': 'sqrt', 'mss': 16}
```

Bagging with DTC:

```
{'criterion': 'entropy', 'md': 14, 'mf': 'sqrt', 'mss': 8}
```

RFC:

```
{'criterion': 'entropy', 'md': 13, 'mss': 10}
```

For xgboost, we varied *subsampling, learning rate,* and *max_depth.* We obtained the following optimal values for xgboost.

```
{'sub': 0.9, 'lr': 0.2, 'md': 8}
```

For Gaussian Naive Bayes, we varied *var_smoothing* and obtained the following optimal value for the same.

```
{'var_smoothing': 6.5793322465756 82e-09}
```

For logistic regression, we varied *solver* and *var_smoothing* and obtained the following optimal value for the same.

```
{'solver': 'newton-cholesky', 'var_smoothing': 1e-09}
```

For KNN, we varied the *algorithm, n_neighbors,* and *weights* and obtained the following optimal value for the same.

```
{'algorithm': 'ball_tree', 'n_neighbors': 5, 'weights': 'distance'}
```

Oversampling

In highly skewed binary classification datasets, where one class has a significantly larger number of samples than the other, the classifier may have a bias towards the majority class, resulting in poor performance on the minority class. Oversampling is a technique that can be used to address this issue.

Oversampling involves generating synthetic samples of the minority class to balance the class distribution. The most commonly used oversampling technique is called Synthetic Minority Over-sampling Technique (SMOTE), which creates synthetic samples by interpolating between the minority class samples.

By oversampling the minority class, the classifier will have more examples to learn from, which can improve its ability to generalize and correctly classify the minority class. This technique can

be particularly useful when the minority class is important or when the cost of misclassifying the minority class is high.

Results

	Recall	Precision	F1-Score	AUPRC
Decision Tree Classifier	0.632911	0.909091	0.746269	0.754504
Random Forest Classifier	0.797468	0.954545	0.868966	0.891137
Bagging	0.822785	0.942029	0.878378	0.886157
XGBoost	0.848101	0.943662	0.893333	0.913626
Logistic Regression	0.670886	0.854839	0.751773	0.806797
Gaussian Naive Bayes	0.392405	0.300971	0.340659	0.366071
K-Nearest Neighbors	0.101266	1.000000	0.183908	0.224079
Voting Classifier	0.835443	0.942857	0.885906	0.905021

Table 1: Results of different models on the original data

	Recall	Precision	F1-Score	AUPRC
Decision Tree Classifier	0.848101	0.770115	0.807229	0.811349
Random Forest Classifier	0.873418	0.896104	0.884615	0.888321
Bagging	0.873418	0.907895	0.890323	0.882367
XGBoost	0.860759	0.894737	0.877419	0.887582

Logistic Regression	0.860759	0.809524	0.834356	0.809632
Gaussian Naive Bayes	0.443038	0.507246	0.472973	0.472973
K-Nearest Neighbors	0.240506	0.413043	0.304000	0.226769
Voting Classifier	0.860759	0.906667	0.883117	0.891937

Table 2: Results of different models on the oversampled data.

Conclusion

We could conclude that oversampling did not change the outcome significantly and the best models before and after oversampling remained the same.
