Batuhan Erden

batuhan.erden@ozu.edu.tr

CS 452/552 – Data Science with Python

Project 3 Report

28.12.2017

Credit Card Fraud Detection

1 Introduction

Throughout history, fraudulent credit card transactions have hurt many people. People have lost their money and could not even find who was responsible for this. As the concept of Machine Learning and Data Science became popular, many solutions that solve this problem have been implemented. First of all, I used the dataset in the kaggle website[1]. The most challenging problem is that the data I have used was unbalanced. Regardless of how unbalanced the data is, there is numerous numbers of solutions that can solve this kind of problems. There are two that I liked the most. The first one is solving it by under sampling the data and the second one is using anomaly detection. I chose the first solution to implement this project. I under-sampled my data such that the number of fraudulent transactions is equal to the number of safe transactions, non-fraudulent transactions. I have tried various numbers of solutions to get a high accuracy and I have used 5-Fold Cross Validation and the Area Under the Precision-Recall Curve (AUPRC) to measure the performance of my dataset. I will describe each and every one of them in the following sections.

[1] https://www.kaggle.com/dalpozz/creditcardfraud

2 Dataset Information

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This is a dataset that contains 284,807 rows and 31 columns in which each row corresponds to a transaction and each column corresponds to a feature and 28 out of 31 columns are the feature, which are the result of a PCA (Principle Component Analysis) due to confidentiality issues and labeled as V1, V2, … V28. The other 3 columns/features are 'Time', 'Amount' and 'Class' which I relabeled it to 'Fraud'. Feature 'Time' contains the information about the seconds elapsed between each transaction and the first transaction in the dataset. Feature 'Amount' is simply the transaction amount whereas feature 'Fraud' is the response variable, which I used as my label data. The feature 'Fraud' is a binary feature that takes 1 if the transaction is a fraudulent transaction and 0 if the transaction is a safe transaction, non-fraudulent transaction. The dataset contains only numerical values so that it can be used to train without vectorizing.

3 Observations

My first approach was to check the number of fraudulent and the number of safe transactions. As can be seen in the *Figure 1*, the dataset is highly unbalanced. The fraudulent transactions account 0.1727% of all transactions. Because of the class imbalance ratio, it is not possible to train this dataset with a straightforward supervised learning. I have done some adjustments to the dataset because there was no way measuring the accuracy correct with the original dataset.

```
492 frauds out of 284807 transactions which is 0.1727% percent of all transactions..
```

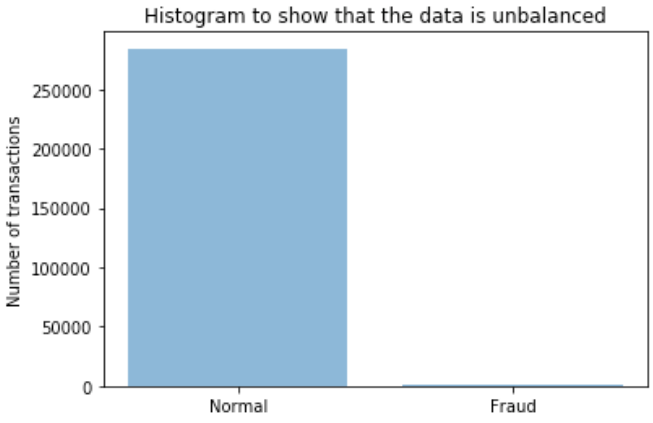

Figure 1: Histogram demonstrating the class imbalance ratio

4 Pre-processing the data

The first thing I did was to ignore the features that have almost no effect on the fraudulent transactions. I have engineered this method after I got some results. My goal was to increase the accuracy. This method was working based on a mean threshold (0.6). For each feature, I observed the fraudulent transaction and took the mean of the rows contains fraudulent transactions. The idea was that if the absolute value of the mean was smaller than or equal to the mean threshold (0.6), it would be ignored and dropped from the dataset. This method had a great improvement on the measured accuracy. While I was implementing this method, I also found out that feature 'Time' had no observable effect on the fraudulent transactions. So, I also decided to ignore/drop that feature/column too. As can be seen below in *Figure 2*, some features (the ones printed at the top) were ignored/dropped and the new data was free of them. The new dataset became like the table in *Figure 2*.

```
Features to be ignored: ['Time', 'V8', 'V13', 'V15', 'V20', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28']
```

	V1	V2	V3	V4	V5	V6	V7	V9	V10	V11	V12	V14	V16	V17	V
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.363787	0.090794	-0.551600	-0.617801	-0.311169	-0.470401	0.207971	0.025
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	-0.255425	-0.166974	1.612727	1.065235	-0.143772	0.463917	-0.114805	-0.183
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	-1.514654	0.207643	0.624501	0.066084	-0.165946	-2.890083	1.109969	-0.121
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	-1.387024	-0.054952	-0.226487	0.178228	-0.287924	-1.059647	-0.684093	1.965
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	0.817739	0.753074	-0.822843	0.538196	-1.119670	-0.451449	-0.237033	-0.038

Figure 2: Features dropped from the dataset (the ones at the top) and the updated dataset (the table at the bottom)

The second thing I did was to check the feature 'Amount'. I tried to cluster them but got a high result that led me to engineer another method. I thought of decreasing the variance between each row. This idea has also made me think about normalizing the feature 'Amount'. After implementing normalizing, I got a good result on the accuracy. This was because normalizing decreased the dimensionality in feature 'Amount'. In *Figure 3*, it is possible the see that feature 'Amount' is normalized.

	Amount
0	0.244964
1	-0.342475
2	1.160686
3	0.140534
4	-0.073403

Figure 3: Normalized amounts

5 Under-sampling

As mentioned earlier, the dataset is highly unbalanced. Due to the class imbalance ratio, the majority of the data would decide the learning output. The first approach I tried was that I feed all data into my classifier and tested the accuracy. The average accuracy was 99.94%. However, this is not correct. The data has 284,315 safe transactions and 492 fraudulent transactions. In the case of using all of them, it would learn all of the 0s but it would not learn the difference between 0s and 1s or it would not learn 1s at all. As a result, the classifications for 1s would not be correct. Yet, the data could still be used to do training. There was a strong way of achieving this.

In order to be able to do the classification correct and learn the features, the gap between the number of fraudulent transactions and safe transactions should not be long. In this project, I decided to keep the ratio between the two transactions 50-50 so that it could learn each of them correctly. To achieve that, I first divided the data into 2 portions: safe and fraudulent. I wanted to keep all of the fraudulent transactions and choose random transactions from safe transactions such that the length of both fraudulent and safe transactions would be the same. I have done my implementation exactly like that. As a result, I had my under-sampled data has the same number of fraudulent and safe transactions. One thing that was left was to shuffle the data because the first 492 elements were 1s and the rest 492 elements were 0s. So, I shuffled the data. The fraudulent and safe transaction values of resultant under-sampled data can be seen in *Figure 4*. *Figure 4* only demonstrates the indexes and the values of the feature 'Fraud'.

	Fraud
40085	1
120505	1
41400	0
223572	1
53794	1
182699	0
246395	0
263080	1
278563	0
245556	1

Figure 4: Feature 'Fraud' of under-sampled data

6 Classifier: Logistic Regression

The classifier used for all of the training operations is Logistic Regression. The reason why I used Logistic Regression was because Logistic Regression is an appropriate regression analysis when the dependent variable is binary. In this problem, our dependent variable is also binary, fraudulent or not. This classifier explains the relationship between one dependent variable and one or multiple independent variables.

Parameters:

- **C:** It is the inverse of regularization strength. I wanted to increase my regularization strength so that I chose **C=0.001** because smaller values specify stronger regularization as in SVMs.
- **Penalty:** It is used to specify the norm used in the regularization. It can either be L1 or L2. After running some tests, I chose the **penalty** parameter to be **L1**.

7 Precision-Recall Curves and Confusion Matrices

For both under-sampled data and actual data, I plotted the precision recall curves and confusion matrices. Needless to say that these are not going to be our final results. Final results were calculated with 5-Fold Cross Validation and Area Under the Precision-Recall Curve.

Precision-Recall Curve: In such cases where the classes are very imbalanced, precision-recall curve is a useful way to measure the success of the prediction. This curve shows the tradeoff between precision and recall. The higher the area under the curve is, the higher the recall and precision is. So, the goal is to maximize this area. The precision is the ability of the classifier not to label a negative sample as positive whereas the recall is the ability of the classifier to find all positive samples. That's why we are interested in recall more in this project. Below are the formulas to calculate them.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

Confusion Matrix: The confusion matrix is often used to describe the performance of a classifier. In our case, it is a 2x2 matrix where rows are actual values and columns are predicted values. Below is the meaning of 2x2 confusion matrix.

[True Negatives, False Positives]
[False Negatives, True Positives]

In both of the following results, the data was split into 4 pieces: *X_train, X_test, y_train, y_test* such that 30% of the data are the test data while the rest are the train data. In addition, Logistic Regression is used as described above.

7.1 Under-sampled data

In this case, I have trained the classifier with under-sampled data and tested it also with under-sampled data. I expected a high scores and accuracies and the results were as I expected. As can be seen in *Figure 5*, the area under the precision-recall curve is high; the average precision-recall score is 95.59%. On the other hand, the results of confusion matrix are also successful but only for 1s. As you can see in the *Figure 5*, only 56.41% of the 0s predicted true. However, since we are only interested in the prediction of 1s (Recall Accuracy), it can be said that it was successful to predict 94.29% of 1s true. To sump up, the results are very good for under-sampled data.

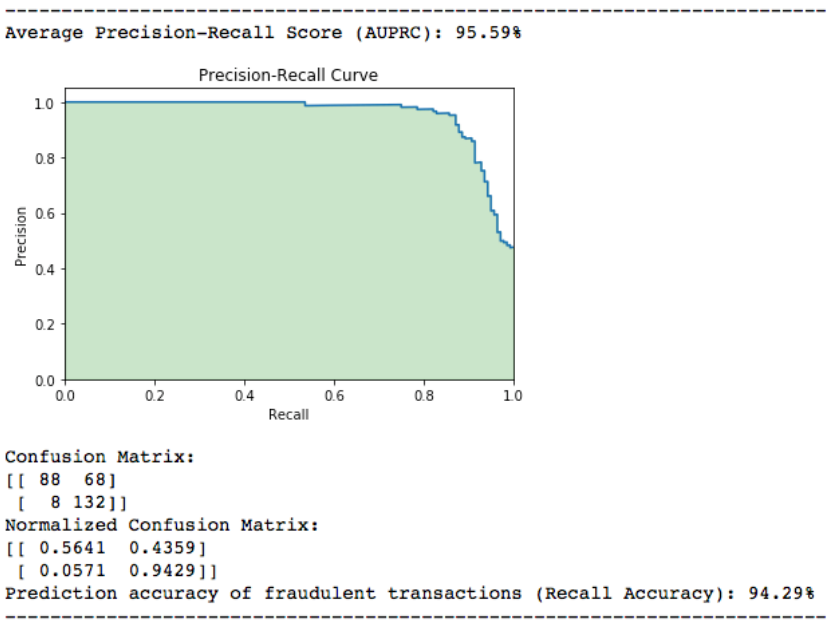

Figure 5: Resultant Precision-Recall Curve and Confusion Matrix of under-sampled data

7.2 Actual data

In this case, I have also trained the classifier with under-sampled data but tested it with the actual data (unbalanced). So, I expected the results would be much worse than the first one. Because I trained my data with 984 rows where the actuals data has 284,807 rows. The results can be seen in *Figure 6*. The area under the precision-recall curve is less than before as expected. The average precision-recall score is 60.91% that is more than overage. The results of the confusion matrix are pretty much different from the other one. Surprisingly, It predicted 1s better. It was 94.29% and it is 94.56% now. Even though it could not classify 0s well (Predicted 57.28% of 0s true), our goal was to implement credit card fraud detection and it looks achieved with this.

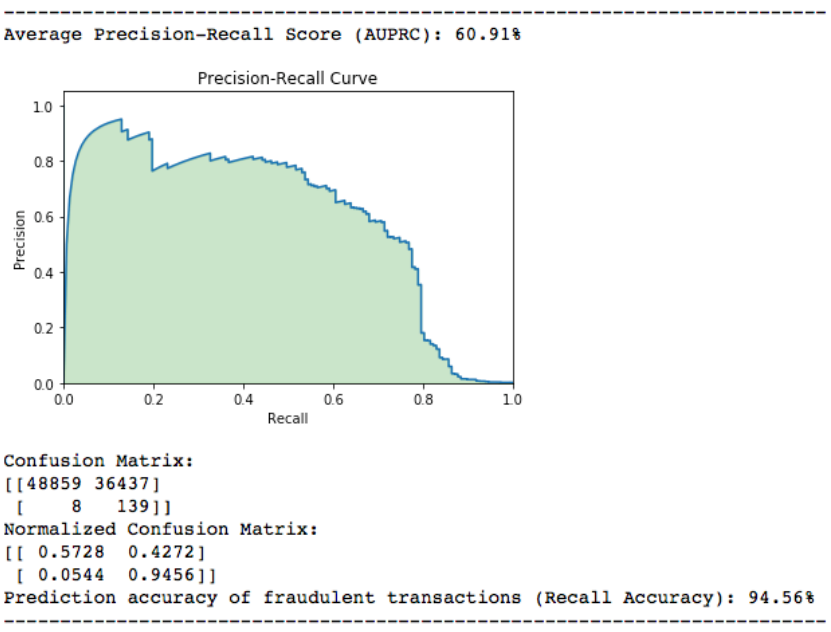

Figure 6: Resultant Precision-Recall Curve and Confusion Matrix of actual data

8 5-Fold Cross Validation and the AUPRC

As described above, I used Logistic Regression to measure the accuracy of the under-sampled data.

K-Fold Cross Validation (k = 5): In this project assignment, I used 5-Fold Cross Validation which basically means that k is **5** in KFold Cross Validation. Classification operation occurs inside the loop which is repeated **5** times and scores are stored at each iteration. In conclusion, the average score accumulated is printed to console.

I used under-sampled data to both train and test. In this part, **decision_function** replaces **predict** because it returned a higher accuracy. For measuring the area under the prediction-recall curve (AUPRC), I used 2 functions from *sklearn's metrics* library. The first one is **roc_auc_score** that measures the area using the **trapezoidal rule**. The second one is **average_precision_score** measures the area without using the **trapezoidal rule**. It basically summarizes a curve as the weighted mean of precisions achieved at each threshold and the increase in recall from the previous threshold that is used as weight.

Trapezoidal Rule: Trapezoidal Rule is a technique for approximating the define integral. It is useful in this case because the area is no different than the integral. I will not go into the details and formulas but trapezoidal rule is basically a rule for measuring the area with the approximation of the definite integral. Furthermore, this rule uses linear interpolation and can be too optimistic in some cases.

Using the methods described above, I generated some results that I didn't think they would be good results. However, they were actually good. In addition, I think choosing **decision_function** over **predict** was a good idea. The final results can be seen in *Figure 7* on the next page.

```
-------------------------------------------------------------
LogisticRegression with 5-Fold Cross Validation
-------------------------------------------------------------
AUPRC found using the trapezoidal rule: 95.39%
AUPRC found without using the trapezoidal rule: 97.24%
AUPRC found using the trapezoidal rule: 96.48%
AUPRC found without using the trapezoidal rule: 97.08%
AUPRC found using the trapezoidal rule: 97.04%
AUPRC found without using the trapezoidal rule: 97.92%
AUPRC found using the trapezoidal rule: 93.60%
AUPRC found without using the trapezoidal rule: 95.14%
AUPRC found using the trapezoidal rule: 92.75%
AUPRC found without using the trapezoidal rule: 95.79%
-------------------------------------------------------------
Average AUPRC found using using the trapezoidal rule: 95.05%
Average AUPRC found using without using the trapezoidal rule: 96.63%
-------------------------------------------------------------
```

Final Result

Average AUPRC (using the trapezoidal rule) < Average AUPRC (without using the trapezoidal rule)

95.05% < 96.63%

Figure 7: Final Results of 5-Fold Cross Validation and AUPRC

9 Conclusion and Discussion

In conclusion, in this project, I have learned to apply numerous numbers of solutions to solve the problem in various numbers of ways. Dealing with unbalanced data was different. Before this project, plotting precision-recall curves and computing confusion matrices was only theoretical. Now I know that they could be used the do classifications. About the results, most of them was surprised me while most of them was exactly as I expected. For example, in *Section 7.2*, I expected the recall accuracy to be less than the one in *Section 7.1* because I only trained my classifier with less than 1000 data and I thought that testing it with more than 200,000 data would give a lower accuracy. Another result that was the inverse of my expectations was in *Section 8* where I used KFold and 2 methods to compute AUPRC. I expected the accuracy of the first method was higher than the accuracy of the second one because the second one was using the trapezoidal rule while the first one was not. Moreover, using trapezoidal rule should have been more optimistic as it is described above. I suppose this is one of the cases where trapezoidal rule was not too optimistic. Regarding of the importance of the data science, I am very happy to tried and solved this problem. In the future, I will also try to solve this problem by using "Anomaly Detection" which I am currently researching.

References

- http://scikit-learn.org
- https://www.kaggle.com