Credit Card Fraud Detection For CS 5824

George Messih, Aritra Majumdar, Mark Wright
Virginia Tech
Blacksburg, VA 24061
george98@vt.edu, aritram21@vt.edu, mtwright21@vt.edu

Abstract

In our project, we worked on the problem of credit card fraud detection. Credit card fraud detection is an in-demand field with lots of application and ability for growth. There are many different ways to create a credit card fraud detection model, in which we discuss three: Random Forest, XGBoost, and neural networks. An underlying theme of every credit card fraud detection model that we touch on is class imbalance. In this final report, we discuss our data, methodology, and final results.

1 Data

Initially we intended to use the IEEE Credit Card Fraud Detection dataset from Kaggle, but upon investigation we discovered the features in this data were not labeled, as they were transformed using PCA. Therefore, we didn't know which feature was money spent, which feature was terminal location vs customer location, etc.. Instead, we used a randomly generated dataset using code from github.com [1].

In the dataset, there is a unique ID for each customer, transaction, and terminal in which the card was used. There are numerical fields indicating transaction amount in dollars and transaction day/time, and fields for average transaction amount over 1 day, 7 day, and 30 day windows for the customer and the terminal where the transaction took place. Lastly, there is a binary feature indicating whether a transaction was legitimate or fraudulent.

2 Methodology

2.1 Class imbalance

A key challenge in any credit card fraud detection model is class imbalance: there are many more cases of legitimate transactions than fraudulent transactions. If the model training is done on the data as it is, it will result in false confidence in your model. Therefore, the distribution of data between the fraudulent class and the non-fraudulent class must be balanced. When we first began our research we attempted to use the SMOTE technique (Synthetic Minority Over-sampling Technique). We ran into issues regarding the boundaries between the nearest neighbors and so for our milestone report we used random over sampling. The results were reasonable, but based on the recommendation from the professor, we used the cost-sensitive learning technique for our final results.

Cost-sensitive learning is a sub-field of machine learning focusing on class imbalance problems. Instead of re-sampling any data, cost-sensitive learning assigns a penalty to each incorrectly classified label in the training data set, and when the model is being trained, it accounts for this penalty. It was shown by Weiss, McCarthy, and Zabar (2007) [2] that cost-sensitive learning had slightly better performance than oversampling in most cases.

2.2 Training/testing split

In time series data, it's common for the relationship among individual features to change over time. These changes in underlying data distributions over time are referred to as concept drift [3]. To account for this we ignored the first two months of our data in our dataset (data began on 4/1/2018) for our models. We used daily credit card data for the two month period 7/15/2018 to 9/16/2018 as our main dataset. The first 70 percent of the data was used for training and validation, and the last 30 percent used for testing.

3 Models and results

Due to the imbalance in the data, the usual accuracy measure is deficient for analyzing model results. Therefore, we use the more relevant metrics of precision and recall, which can be computed from the model's confusion matrix. Further, we look at the full range of precision and recall values across the full range of data for each threshold t using the Precision-Recall (PR) and Receiving Operating Characteristic (ROC) curves. We summarize the whole PR curve using the average precision (AP) and the whole ROC curve using the area under the curve (AUC). Typically there is a trade-off between precision and recall. One metric captures useful information the other doesn't (precision accounts for false positives while recall accounts for false negatives) and using just one can lead to valuable information being lost by the modeler. Given this trade-off, we consider both the AP and AUC of each model in tandem when assessing results.

3.1 Random forest

We employed three separate Random Forest models with different depths to see which depth performed the best. For each random forest, we used entropy as the purity measure. We accounted for the class imbalance as mentioned above using a module from sklearn titled class_weight so the model would assign penalties to each misclassified data point. Upon comparing results we determined the random forest with depth of 10 performed the best.

The ROC curve is convex, and the values are consistently greater than 0.5 for both the True Positive Rate (TPR) and False Positive Rate (FPR). The fact that TPR is much higher than FPR shows the fact that the model can detect frauds well overall. We can see the AUC is a healthy 0.853. The Precision-Recall curve mirrors the ROC curve, and the average precision is 0.504. It is observed that the precision drops sharply when the recall reaches approximately 0.75. The trade-off between precision and recall means we would need to focus on the recall values more, as the recall is a measure of what percentage of actual frauds the classifier successfully identifies.

Table 1: Random Forest Confusion Matrix

	Pred Non Fraud	Pred Fraud	Total
Actual Non Fraud	$181,692$	492	$182,184$
Actual Fraud	485	1185	1670
Total	$182,177$	1677	$183,854$

From Table 1 it is observed that out of the 1670 actual fraud cases, 1185 of them were accurately identified as frauds while around 492 of the non-fraudulent data were misclassified as frauds. This means that Random Forest, while being able to identify fraudulent transactions relatively well, is still not very effective as it misclassifies non-frauds as frauds as well to compensate for the class imbalance.

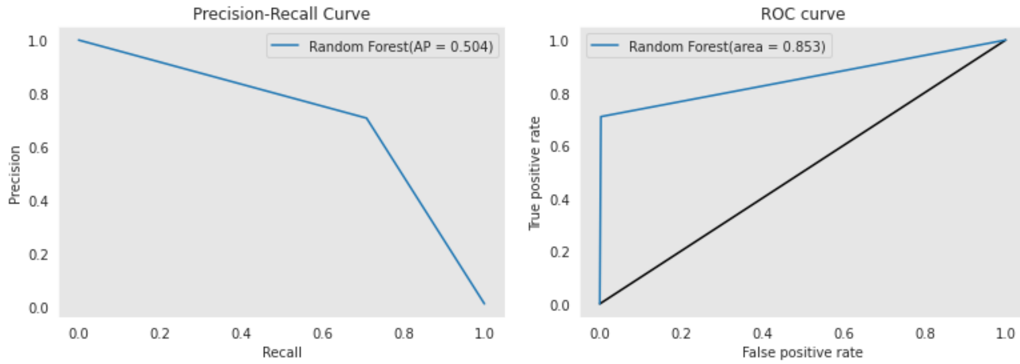

Figure 1: Random Forest Precision-Recall and ROC curves.

3.2 XGBoost

We also employed three separate XGBoost models, but each with separate learning rates. We tried learning rates of 0.1, 0.2, and 0.3 to see which performed the best. Again, we accounted for the class imbalance using the class_weight module from sklearn. Upon viewing the results we determined the model using a learning rate of 0.1 performed the best.

Similar to the Random Forest model, the ROC curve is convex and the values are consistently greater than 0.5 for both the TPR and FPR. Similarly for the PR curve, the precision is consistently higher for each recall value for the XGBoost model than it was for the Random Forest model, and the average precision is considerably higher at 0.677. The precision starts to drop considerably when recall is around 0.8, which is a good sign as this allows for higher recall values compared to Random Forest, and consequently, higher accuracy of classifying frauds correctly. Though the AUC for XGBoost is slightly lower than the random forest, the considerable improvement in the average precision tells us that this model performs better than the Random Forest.

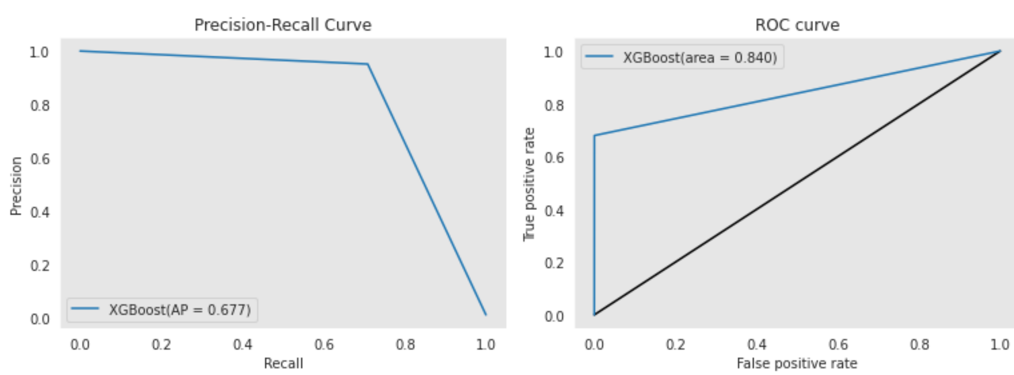

Figure 2: XGBoost Precision-Recall and ROC curves.

The confusion matrix in Table 2 provides additional insight into the performance of the XGBoost model. In contrast to Random Forest (Table 1), the XGBoost does not misclassify non-frauds as frauds as much. However, the accuracy of correctly identifying frauds is still similar to the Random Forest model.

Table 2: XGBoost Confusion Matrix

	Pred Non Fraud	Pred Fraud	Total
Actual Non Fraud	182,171	50	182,221
Actual Fraud	523	1110	1633
Total	182,694	1160	183,854

3.3 Neural Networks

For the novelty of our project, we took into account the sequential nature of the data for training our neural network. Most fraud detection models don't look at an individual customer's sequence of transactions in model training, i.e. they use the dataset as a whole usually. We accounted for this sequence of transactions at the customer level by using a Long Short-Term Memory Neural Network (LSTM) on the dataset. Each row of the data represented a customer, and the dataset features were ordered according to the dates of transactions to keep the sequence of transactions in mind. The features were then scaled using the standard scaler-from-preprocessing submodule from the sklearn module. The LSTM model had 256 hidden neurons and was trained for 20 epochs. The last hidden representation of the LSTM was passed through two linear layers to eventually provide one output with a sigmoid activation to generate the probabilities for the training sequences to be fraudulent or non-fraudulent. For training, Adam optimizer was used with a learning rate of 0.0001 paired with Binary Cross Entropy loss for calculating backpropagation.

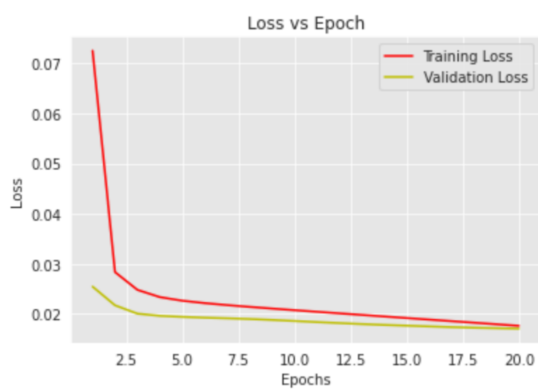

Figure 3: Training and Validation losses vs Epoch.

From Fig 3 it is observed that the training loss and validation loss convergs at the 20th epoch which means that the model has not overfit the dataset. To get the predictions, a threshold value of 0.5 was used, i.e., a probability greater than 0.5 would result in a fraudulent class. To increase the efficacy of the model, lowering the threshold to 0.4 or 0.3 was considered, but that allowed for too many non-fraudulent transactions to be misclassified as fraudulent, which would not be helpful for our problem.

Table 3: LSTM Confusion Matrix

	Pred Non Fraud	Pred Fraud	Total
Actual Non Fraud	75,662	61	75,723
Actual Fraud	251	415	666
Total	75,913	476	76,389

The confusion matrix, the PR Curve, and the ROC Curve have also been provided in Table 3 and Fig 4, respectively. Note that because of the grouping by customer for the LSTM described earlier, there are fewer data points in this confusion matrix than there were for the Random Forest and XGBoost's matrices. The LSTM confusion matrix shows that almost 33 percent of the fraudulent transactions were incorrectly classified as non-fraudulent by the LSTM model. It is interesting to note that this is consistent with the rest of the models (Tables 1 and 2).

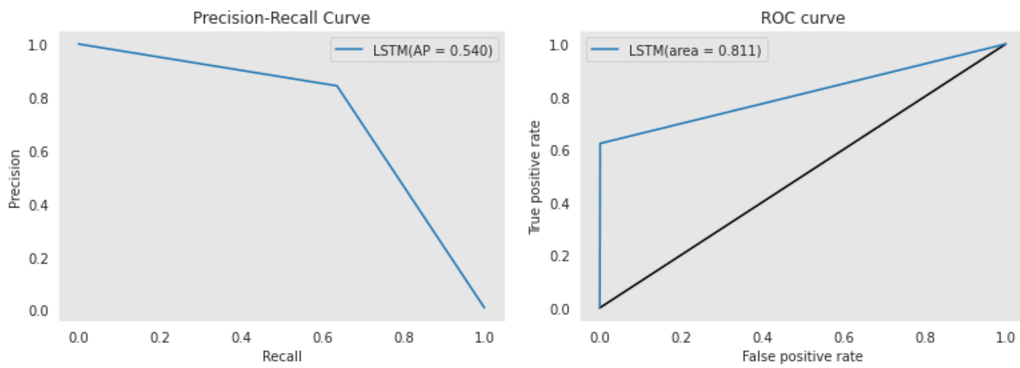

Figure 4: LSTM Precision-Recall and ROC curves.

From Fig 4 it is observed that the LSTM model also performs reasonably well on the dataset. An average precision of 0.54 percent is higher than that of the Random Forest classifier, and an AUC of 0.811 percent is reasonably good. The ROC curve is convex, and the values are consistently greater than 0.5 for both True Positive Rate (TPR) and False Positive Rate (FPR).

4 Conclusion

Even though each model considers some amount of non-fraudulent transactions as fraud, in this domain detecting fraudulent transactions correctly is more important than determining whether a transaction is non-fraudulent. The obstacle of non-fraudulent transactions being incorrectly labeled as fraudulent can be overcome by confirming via real world interactions. Therefore, we conclude each model can detect frauds well, but the XGBoost model did the best job predicting frauds as it had significantly higher average precision while it's AUC was still comparable to the LSTM and Random Forest models. Behind XGBoost, we contend the Random Forest performed second best, followed by LSTM.

In hopes of optimizing our models, we drew inspiration from a method widely used in research: Generative Adversarial Networks, or GANs. The GANs framework is an iterative min-max simulation/game that involves 2 players, a Generator model (G) and a Discriminator model (D). The G model aims to generate new fraudulent transactions in each round of the game, with the goal of maximizing the number of fraudulent transactions that are not detected/correctly classified by the D model, while the D model aims to minimize that same number of frauds. By retraining both models after each round, the goal is that both models become more efficient at each of their tasks by facing a more advanced opponent in each new round. Another advantage is to simulate how our models would perform against the repeated attacks faced in the real world against more advanced adversaries. Unfortunately, this is a complex topic and we were unable to improve our models using GANs given our time frame, but we contend this an exciting area for further research.

5 Contributions

Mark generated the data and researched the cost-sensitive learning methodology, Aritra created the models in Python using Scikit-learn and Pytorch, and George tested methods to deal with class imbalance, created the GAN framework, and made the base PowerPoint for our Spotlight Presentation.

References

[1] Le Borgne, Yann-Aël and Siblini, Wissam and Lebichot, Bertrand and Bontempi, Gianluca. (2022). *Reproducible Machine Learning for Credit Card Fraud Detection - Practical Handbook (https://github.com/Fraud-Detection-Handbook/fraud-detection-handbook).* Université Libre de Bruxelles.

[2] Weiss, G. M., McCarthy, K., & Zabar, B. (2007). *Cost-sensitive learning vs. sampling: Which is best for handling unbalanced classes with unequal error costs?*. Dmin, 7(35-41), 24.

[3] Žliobaitė, I., Pechenizkiy, M., & Gama, J. (2016). *An overview of concept drift applications.* Big data analysis: new algorithms for a new society, 2, 91-114.

[4] Saito, M. (2015). *The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets.* PLOS ONE, 10, 1-21.

[5] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). *Generative adversarial nets. Advances in neural information processing systems*, 27.

[6] Chakraborty, A., Alam, M., Dey, V., Chattopadhyay, A., Mukhopadhyay, D. (2018). *Adversarial attacks and defences: A survey.* arXiv preprint arXiv:1810.00069.