Machine learning - Fraud Transaction Detection

Venkata Vamshi Krishna Gunji

Email: venkatavamshi.gunji@gmail.com or gvvamshikrishna@gmail.com

LinkedIn: https://www.linkedin.com/in/venkata-gunji/
GitHub: https://github.com/vamshigunji183

Overview

Problem: Fraudulent transaction detection

Challenges: Imbalance data, time series transactional data

Goal: Well performing model with low FN misclassification

Algorithms: Neural networks, K-Means, Logistic Regression

Concepts: PCA, Oversampling, hyperparameter tuning, Confusion matrix, Gaussian Noise

1 Problem and Goals

Goal is to implement 3 different machine learning models in order to classify, to the highest possible degree of accuracy, credit card fraud from a dataset gathered in Europe in 2 days in September 2013. After initial data exploration, discovered to implement a logistic regression model, a k-means clustering model, and a neural network.

Some challenges observed from the start were the huge imbalance in the dataset: frauds only account for 0.172% of fraud transactions.

In this case, it is much worse to have false negatives than false positives in our predictions because false negatives mean that someone gets away with credit card fraud. False positives, on the other hand, merely cause a complication and possible hassle when a cardholder must verify that they did, in fact, complete said transaction (and not a thief).

2 Data Processing

One challenging aspect of the dataset was that there were 30 features, but in order to protect confidentiality, all but 28 of them had been PCA transformed and had unknown labels. The known, non-transformed features were 'Time', which measures the seconds between the transaction and the first transaction in the 2-day time period, and 'Amount', which is the cost of the transaction, presumably in Euros.

In order to offset the imbalance in the dataset, we oversampled the fraud (class = 1) portion of the data, adding Gaussian noise to each row.

3 Data Modeling

The 3 models we used were a fully connected neural network, K-means, and logistic regression.

In terms of the neural network, performing principal component analysis on the oversampled data before splitting it into training and test sets resulted in a jump from 50% accuracy to 94.56% accuracy. Before PCA, nothing we tried was able to push the accuracy past 50%. After this step, however, adjustments to the number of layers, activation functions, and neurons in each layer did not do much to change the accuracy, which hovered at just below 95%. Furthermore, choosing only 2 neurons for the first dense layer (called a "bottleneck effect") forced the model to really reduce to only the most necessary features and decrease the likelihood of over fitting.

For the K-means model, principal component analysis was used to reduce the dimensionality of the data from 31 to 2. Only the two features with the most variance were used to train the model. The model was set to have 2 clusters, 0 being non-fraud and 1 being fraud. We also experimented with different values for the hyperparameters, but they all produced similar results. Changing the dimensionality of the data (reducing it to more dimensions than 2) also made little difference on the final values.

The last model we used was logistic regression, as it was a good candidate for binary classification. On each logistic regression model we trained, we made the constant C to be $1 * 10^{-5}$. We trained three different configurations: a vanilla logistic regression with no preprocessing whatsoever, a logistic regression with oversampling on the scarce fraudulent data points and data scaling, and a logistic regression with balanced weights for each class (which greatly helped the data unbalance).

4 Comparison of Models

Logistic regression outperformed both the K-means and neural network. We believe that it is because of how the decision boundary changed with the class weights features. The neural network was next, and K-means performed the poorest. We believe this is due to the fact that clustering relies entirely on the similarities and differences of features of the dataset. Since fraud transactions can look very similar to regular transactions, it is difficult to put them into a separate group based on features alone.

5 Discussion of Results

While the neural network had a high accuracy, its biggest pitfall was that within its 5.44% inaccuracy rate, 84.64% were false negatives. The fact that this neural network missed 4.60% of frauds is enough to make this model infeasible compared to the other methods, where the false negative rate was much lower. Interestingly, a switch from a sigmoid to tanh activation function reduced the false negative rate by about 1%.

The K-means clustering model produced a low accuracy of 54.27%. Of the wrongly predicted transactions, 99.75% were false positives, giving only 0.24% false negatives, or 0.11% of the validation set. However, the false negative rate was only so low due to the extremely low proportion of frauds in the dataset. In reality, 112 of the 176 frauds were misclassified as non-frauds, giving this a true accuracy rate of 36.36%. Therefore, K-means would not be the preferred model for this dataset, as it did not correctly predict frauds and it also produced a lot of false positives.

The logistic regression gave us the best results. The vanilla logistic regression gave us a great accuracy rate of 99.88%, with 0.079% of the validation set being false negatives (or 0.49% of the number of misclassifications). The logistic regression with oversampling gave us an interesting result, as they performed worse than the vanilla logistic regression. The accuracy was 98.01%, with 1.56% of the validation set being false negatives (or 3.12% of the misclassifications). Lastly, the logistic regression with balanced weights achieved the best results: although the accuracy was 97.5%, just 0.011% of the validation set resulted in false negatives (or 0.44% of the misclassifications).

6 Conclusion

Further, I would have liked to research and experiment more with adjustments to the layers of the neural network. During the research, it was clear that some people used very sophisticated techniques to try to optimize these parameters, while our method was mostly guess and check. I found that a random forest model was often the best classifier, so implementing that would be another next step. Additionally, planning to implement an autoencoder or try our hand at an SVM to see how that performed.