Credit card fraud detection using Machine Learning techniques

Anzhel Abgaryan
Yerevan State University, Yerevan, Armenia

Abstract

Credit card fraud is an inclusive term for fraud committed using a payment card, such as a credit card or debit card. The purpose may be to obtain goods or services, or to make payment to another account which is controlled by a criminal. The Payment Card Industry Data Security Standard (PCI DSS) is the data security standard created to help businesses process card payments securely and reduce card fraud.

1. Introduction

Nowadays payment operators and financial institutions are mostly relying on different risk management tools to identify and detect fraudulent transactions. Those tools are fraud detection systems, which are working mostly based on using machine learning algorithms. In this project, I used some popular machine learning models, including logistic regression, decision trees, support vector classification and k-nearest neighbor classification, and a real-life dataset from Kaggle. The main challenge of this project is that in real life, the fraud transactions usually represent a very small percent of all the transactions. The data, which was used for this project is containing only 0.2% of fraudulent transactions. This means that the dataset is heavily imbalanced, and standard approaches cannot be a solution for this case.

2. Dataset and features

The dataset contains 31 variables and about 300,000 rows of credit card transactions. Table 1 provides detailed description of the dataset and variables, which comes from Kaggle. As you can see another challenge of this dataset is that 28 out of the total 31 variables are the result of dimensionality reduction which was done to protect sensitive information.

Table 1: Dataset description

Variables	Description
Time	Number of seconds elapsed between this transactions and the first transaction in the dataset
V1 || V28	May be result of a PCA Dimensionality reduction to protect user identities and sensitive features
Amount	Transaction amount
Class	1 for fraudulent transactions, 0 otherwise

Preliminary analysis reveals several interesting features about the data. Table 2 shows the split of fraudulent and legitimate transactions. Only 492 transactions, or less than 0.2 percent, are fraudulent, highlighting a stark class imbalance.

Table 2: Distribution of the target variable

	Number	Percentage
All transactions	284,807	100
Fraud	492	0.173
Nonfraud	284,135	99.827

Table 3 shows that fraudulent transactions on average have greater transaction value than legitimate transactions.

Table 3: Average transaction value by fraudulent status

	Total	Average
All transactions	25,162,590	88.35
Fraud	60,128	88.29
Nonfraud	25,102,462	122.21

A correlation analysis (Figure 1) reveals that the class labels (the very last column) are correlated with several of the PCA variables. But, the PCA variables are uncorrelated with each other, consistent with the fact that they are orthogonal and have been standardized to have zero mean.

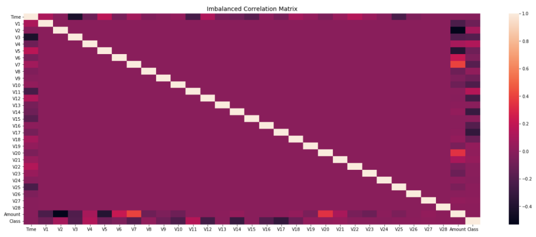
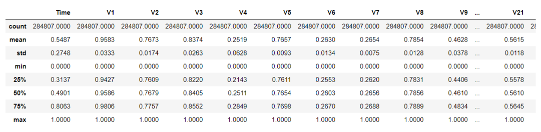


Figure 1. Correlation Matrix of initial dataset.

In order to boost convergence, i.e. make models converge faster I applied min-max normalization to the all features, except the target variable. This is the min-max normalization equation:

$$z_i = \frac{x_i - min(x)}{max(x) - min(x)}$$

With the help of it the dataset was scaled, and all values are from 0 to 1.



	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21
count	284807.0000	284807.0000	284807.0000	284807.0000	284807.0000	284807.0000	284807.0000	284807.0000	284807.0000	284807.0000	...	284807.0000
mean	0.5487	0.9583	0.7673	0.8374	0.2519	0.7657	0.2630	0.2654	0.7854	0.4628	...	0.5615
std	0.2748	0.0333	0.0174	0.0263	0.0628	0.0093	0.0134	0.0075	0.0128	0.0378	...	0.0118
min	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	...	0.0000
25%	0.3137	0.9427	0.7609	0.8220	0.2143	0.7611	0.2553	0.2620	0.7831	0.4406	...	0.5578
50%	0.4901	0.9586	0.7679	0.8405	0.2511	0.7654	0.2603	0.2656	0.7856	0.4610	...	0.5610
75%	0.8063	0.9806	0.7757	0.8552	0.2849	0.7698	0.2670	0.2688	0.7889	0.4834	...	0.5645
max	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	...	1.0000

Figure 2. Statistical overview of the scaled data.

3. Methods

In this section I will tell about sampling techniques and machine learning algorithms that were used. Also, in order to optimize given models and determine the optimal values for them, I used Grid search hyperparameter tuning.

3.1. Sampling method

As I mentioned previously, the dataset is heavily imbalanced (see Figure 3), and common approaches are not suitable.

Class imbalance in Machine Learning is a wide area of research. Various techniques are deployed to address this problem, including oversampling the minority class, undersampling the majority class, generating synthetic samples of the minority class, and adjusting the relative cost of misclassifying the minority and majority class. as a sampling technique I chose undersampling, as oversampling can lead to overfitting, and added noise from synthetic data generation can reduce predictive performance. Also, the result of undersampling the dataset is much more smaller dataset, so the overall research won't require big computational power and won't lead to overflowing and memory errors.

Figure 4 describes dataset distribution after the undersampling.

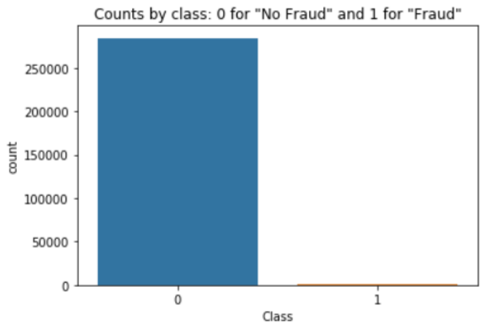


Figure 3. Initial target class distribution.

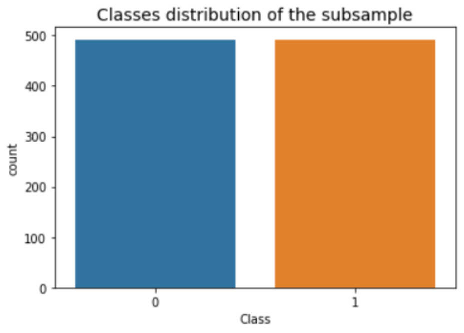


Figure 4. Target class distribution after undersampling.

3.2. Logistic Regression

Logistic regression is a widely used discriminative learning algorithm that uses a hypothesis of the form:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

In its simplistic form, logistic regression fits a linear decision boundary in the feature space to model a binary dependent variable. Parameters are fit by maximum likelihood, where the log likelihood is given by

$$\ell(\theta) = \sum_{i=1}^{n} y^{(i)} \log h\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h\left(x^{(i)}\right)\right)$$

For this model GridSearch was given following parameters:

”penalty”: [’l1’, ’l2’],
”C”: [0.001, 0.01, 0.1, 1, 10, 100, 1000],
where ”penalty” specify the norms used in the penalization, and ”C” represents the inverse of regularization strength. According to the GridSearch, best model was with penalization with l2 norm, and
C = 1000. This is also called Ridge regularization.

3.3. K-nearest Neighbor

KNN is a supervised machine learning algorithm used in Machine Learning for regression and classification problem. KNN algorithms use data and classify new data points based on similarity measures (e.g. distance function). Classification is done by a majority vote to its neighbors. The data is assigned to the class which has the nearest neighbors. As you increase the number of nearest neighbors, the value of k, accuracy might increase.

The k-nearest-neighbor classifier is commonly based on the Euclidean distance between a test sample and the specified training samples. Let xi be an input sample with p features $(x_{i1}, x_{i2}, \ldots, x_{ip})$, n be the total number of input samples (i=1,2,,n) and p the total number of features (j=1,2,,p). The Euclidean distance between sample xi and xl (l=1,2,,n) is defined as

$$d\left(\mathbf{x}_i, \mathbf{x}_l\right) = \sqrt{\left(x_{i1} - x_{l1}\right)^2 + \cdots + \left(x_{ip} - x_{lp}\right)^2}$$

For this model GridSearch was given following parameters: ”n_neighbors”: [2, 3, 4, 5] ,

’algorithm’: [’auto’, ’ball_tree’, ’kd_tree’, ’brute’],

where ”n_neighbors” represent number of neighbors, and ”algorithm” represents the algorithm used to compute the nearest neighbors: ball_tree will use BallTree, kd_tree will use KDTree, brute will use a brute-force search and auto will attempt to decide the most appropriate algorithm based on the values passed to fit method.

According to the GridSearch, best model was with parameters ”auto” and 4 neighbors.

3.4. Decision Tree

Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

For this model GridSearch was given following parameters:
”criterion”: [”gini”, ”entropy”],
”max_depth”: [2, 3, 4],
”min_samples_leaf”: [5, 6, 7],
where ”criterion” represent The function to measure the quality of a split. The definition of entropy is:

$$I_H(t) = -\sum_{i=1}^{c} p(i \mid t) \log_2 p(i \mid t)$$

where, $p(i \mid t)$ is the proportion of the samples that belongs to class i for a particular node t. The entropy is therefore 0 if all samples at a node belong to the same class, and the entropy is maximal if we have a uniform class distribution.

The Gini impurity can be understood as a criterion to minimize the probability of misclassification:

$$I_G(t) = \sum_{i=1}^{c} p(i \mid t)(1 - p(i \mid t)) = 1 - \sum_{i=1}^{c} p(i \mid t)^2$$

The ”max_depth” represents maximum depth of the tree, and ”min_samples_leaf” - the minimum number of samples required to be at a leaf node.

According to the GridSearch, best model was with parameters ”entropy”, maximum depth of 4 and 3 minimum required samples.

3.5. Support Vector Classifier

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side. In SVMs, our optimization objective is to maximize the margin. The margin is defined as the distance between the separating hyperplane (decision boundary) and the training samples that are closest to this hyperplane, which are the so-called support vectors. The hypothesis function h is defined as:

$$h\left(x_i\right) = \begin{cases} +1 & if w \cdot x + b \geq 0 \\ -1 & if w \cdot x + b < 0 \end{cases}$$

Computing the SVM classifier amounts to minimizing an expression of the following form:

$$\left[\frac{1}{n} \sum_{i=1}^{n} \max\left(0, 1 - y_i\left(w \cdot x_i - b\right)\right)\right] + \lambda\|w\|^2$$

For this model GridSearch was given following parameters:
'C': [0.5, 0.7, 0.9, 1],
'kernel': ['rbf', 'poly', 'sigmoid', 'linear'],
where "C" represent regularization parameter, and "kernel" represents the kernel type to be used in the algorithm: linear, poly, rbf, sigmoid, precomputed or a callable.

According to the GridSearch, best model was with parameters 'C' = 0.9, and 'poly' for kernel, which means a polynomial was used to find the hyperplane to split the data.

4. Results and conclusion

There are many metrics to evaluate machine learning algorithms and models. The most popular metric so far is the accuracy, but again, it is not suitable for this case, due to the class imbalance. For example, if we'd always stated that the transaction is not fraudulent, the accuracy would be 99,7%, which is considered as excellent coefficient for any metric. Also, in this case, if a fraudulent transaction (Actual Positive) is predicted as non-fraudulent (Predicted Negative), the consequence can be very bad for the financial institution. So we mostly care about False Negative values. That's why we should use ROC AUC score as an evaluation metric for our models.

Figure 5, Figure 6, Figure 7 and **Figure 8** show the confusion matrices for Desicion tree, KNN, logistic regression and SVC respectively.

Looking at the **Table 3**, we can tell that the best model according to ROC AUC score is Logistic Regression with Ridge regularization.

Figure 9 shows ROC AUC scores of the models. It was calculated with scikit-learn library.

Figure 10, Figure 11, Figure 12 and **Figure 13** show classification report results for KNN, logistic regression, Decision tree and SVC respectively.

Table 3: ROC AUC scores for the models

Logistic Regression	0.9729493891797556
K Nearest Neighbors	0.9115150927541853
Support Vector Classifier	0.9707258742162757
Decision Tree Classifier	0.91719669058238

5. Future work

As technology changes, it becomes difficult to track the behavior and pattern of fraudulent transactions. Preventing known and unknown fraud in real-time is not easy but it is feasible. In future might be done oversampling and synthetic data generation methods, I think in case of properly tuned algorithms, they can lead to superior predictive performance in the face of class imbalance. Given that payments fraud is constantly evolving, future areas of work might include the application of reinforcement learning to a real-time data stream. Although fraudsters will never retire, machine learning algorithms can give them a run for their money.

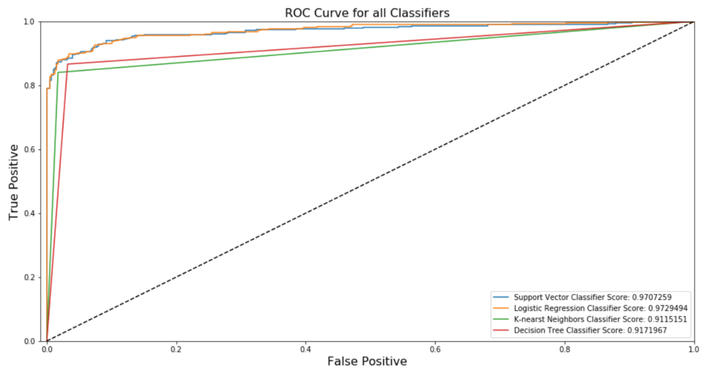


Figure 5. AUC ROC plot.

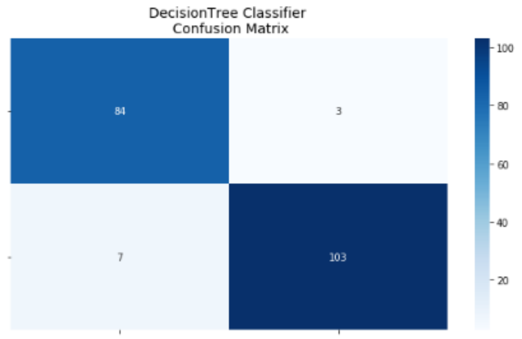


Figure 6. Confusion Matrix for decision tree.

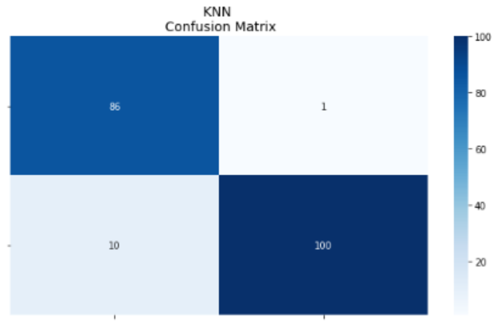


Figure 7. Confusion Matrix for KNN.

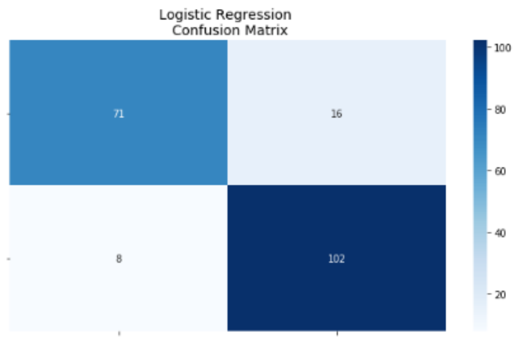


Figure 8. Confusion Matrix for logistic regression.

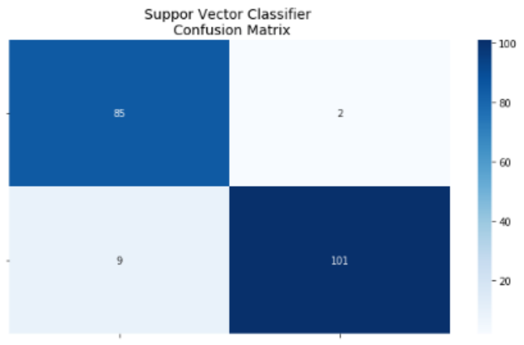
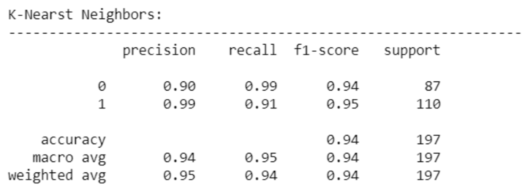


Figure 9. Confusion Matrix for SVC.

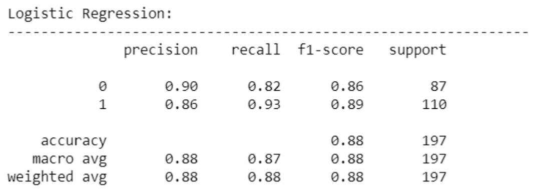


```
K-Nearst Neighbors:
-----------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.90      0.99      0.94        87
           1       0.99      0.91      0.95       110

    accuracy                           0.94       197
   macro avg       0.94      0.95      0.94       197
weighted avg       0.95      0.94      0.94       197
```

Figure 10. Classification report results for KNN.

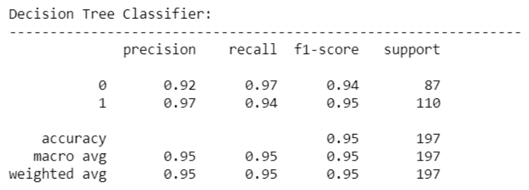


```
Logistic Regression:
-----------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.90      0.82      0.86        87
           1       0.86      0.93      0.89       110

    accuracy                           0.88       197
   macro avg       0.88      0.87      0.88       197
weighted avg       0.88      0.88      0.88       197
```

Figure 11. Classification report results for logistic regression.

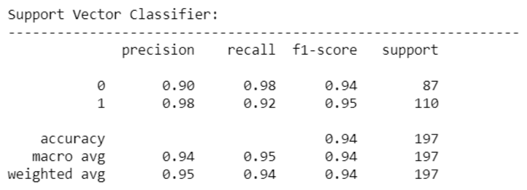


```
Decision Tree Classifier:
-----------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.92      0.97      0.94        87
           1       0.97      0.94      0.95       110

    accuracy                           0.95       197
   macro avg       0.95      0.95      0.95       197
weighted avg       0.95      0.95      0.95       197
```

Figure 12. Classification report results for Desicion tree.



```
Support Vector Classifier:
-----------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.90      0.98      0.94        87
           1       0.98      0.92      0.95       110

    accuracy                           0.94       197
   macro avg       0.94      0.95      0.94       197
weighted avg       0.95      0.94      0.94       197
```

Figure 13. Classification report results for SVC.

6. References

- Dataset - https://www.kaggle.com/mlg-ulb/creditcardfraud
- Code available on my github - https://github.com/anzhel-abgaryan/ASDS_ML
- Model parameter definitions - https://scikit-learn.org/
- Pattern Recognition and Machine Learning (Information Science and Statistics), Christopher M. Bishop
- Python: Deeper Insights into Machine Learning, Sebastian Raschka

References