Final Report

Comparison of supervised and unsupervised learning algorithms in credit card fraud detection

Andrea Koltai
********@gatech.edu
GTID: *********

****
********@gatech.edu
GTID: *********

12/08/2020

1 Motivation and problem statement

Anomaly detection (a subfield of which is also known as outlier detection) refers to a data analysis technique that aims to find the very rare observations that are different from the normal or the vast majority of the observations. This method is widely used in different scientific areas, for example: in healthcare identifying abnormal cells to aid establishing a diagnosis, in cybersecurity identifying network or server attacks/breaches, in finance for identifying high risk operational events such as fraudulent credit card transactions. The general characteristic of anomalous activity is that it occurs with a very low probability but might be of very high negative impact and thus can cause substantial cost and/or liability for organizations and stakeholders involved. Inspired by this, in our project we will attempt to identify fraudulent credit card transactions.

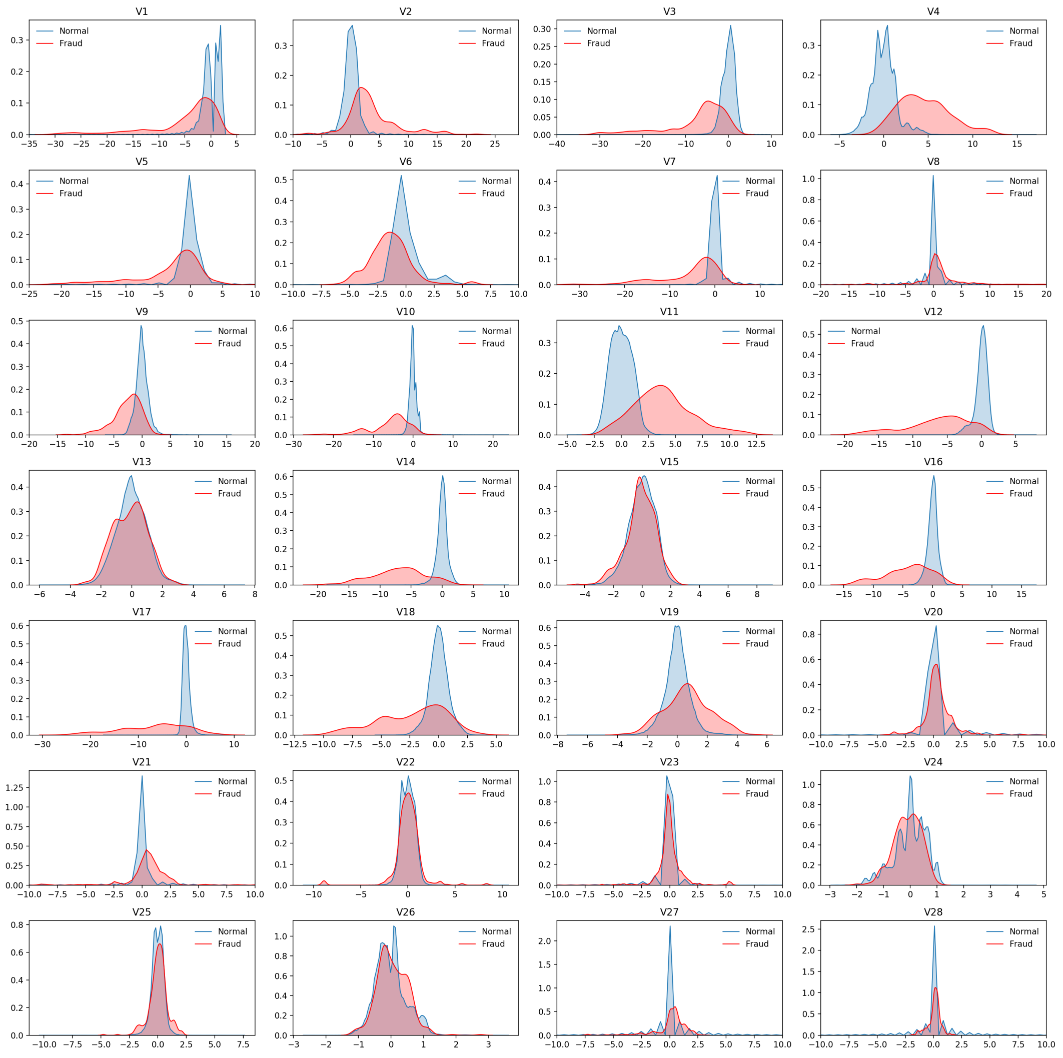
2 Data source

For our project we will use the Credit Card Fraud dataset available on Kaggle [1]. The dataset consists of 284,807 credit card transactions occurred in the span of two days in September 2013 initiated by European cardholder customers of an unnamed bank/credit card processor. The dataset is extremely imbalanced, contains only 472 transactions labeled as fraud, which is a mere 0.172% of all the observations. The data set contains 30 numerical variables: the 'Time' and the 'Amount' of transactions and additional 28 numerical features ('V1', 'V2', ... 'V28') which are the result of a PCA transformation. The 'Time' variable describes the elapsed time in seconds between each transaction and the first transaction in the dataset. The labels are represented in the 'Class' response variable, which equals to '1' in case of fraudulent transactions otherwise it is '0'. Due to confidentiality issues the cardholder didn't provide the original features. Although the dataset can be considered as 'ready for use' and we do not have to handle correlation within the features, it also limits the analysts options for feature engineering to just keeping or dropping certain variables.

3 Exploratory data analysis

As part of our analysis we estimated the density for every feature separately for normal and fraudulent transaction, as shown in Figure 1. The kernel density estimation can help us to identify the most important/predictive features. Features with very similar distributions for fraud and normal transactions (for example 'V15' or 'V22') will probably less helpful to identify anomalies, but features with completely different distributions (for example 'V17' and 'V14') can be more effective in finding the fraudulent transactions.

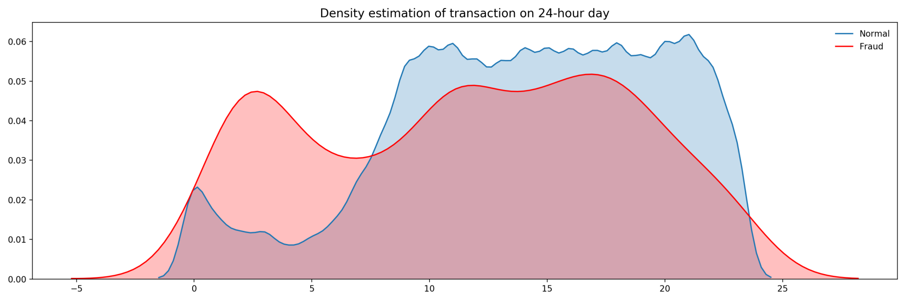
Figure 1: Kernel Density estimation of V1-V28 features



We can also notice interesting pattern in the distribution of transactions in time as shown in Figure 2. There are typical time periods when the normal transactions occur and there are time periods in which the occurrence of normal transactions are much lower, however the fraudulent transactions shows us a much smoother distribution in time. This suggested us create a variable based on in which hour the transaction happened within a day.

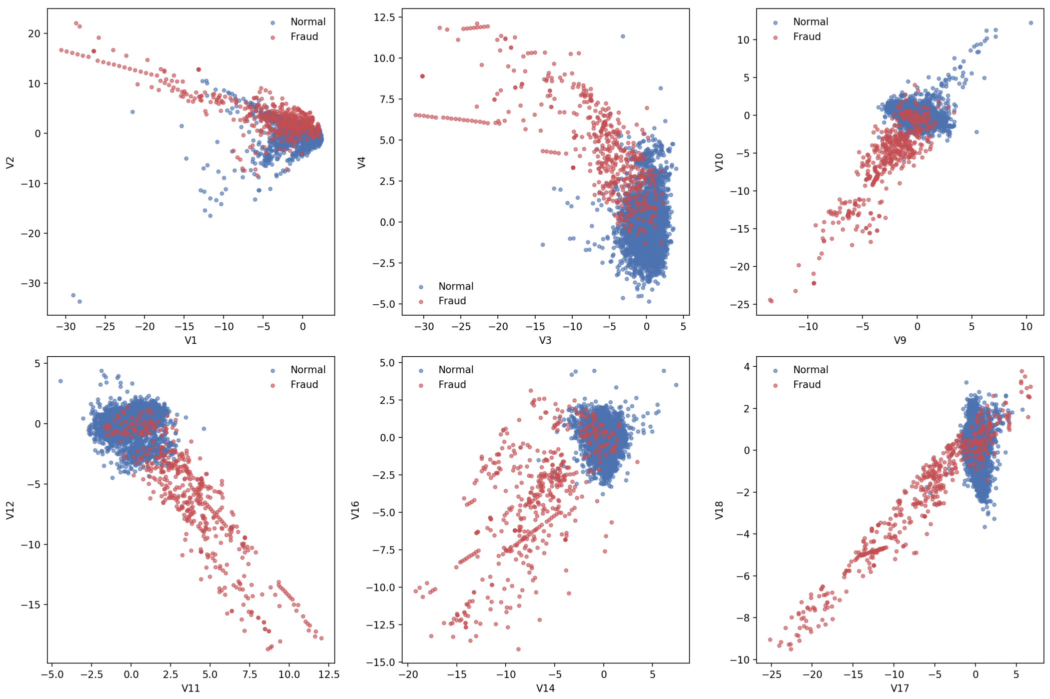
The Figure 3 illustrates the scatter plots of few selected variables (selection was based on the KDE estimation) separately for fraudulent and normal transactions. We can see that some

Figure 2: Distribution of the normal and fraudulent transactions within a day



pair of the PCA components show substantial variance in fraudulent transactions and indicates to become predictive in anomaly detection. We can also notice the substantial correlation of these PCA components but only in respect of the fraudulent transactions. We can see that the majority of the fraudulent transactions can be separated linearly and this indicates that even linear models can perform quite well solving our outlier detection problem.

Figure 3: 2D scatter plot of selected PCA components



4 Methodology

The goal of our project is to identify the fraudulent transactions among a large set of credit card transactions and to compare the effectiveness of the supervised and unsupervised machine learning algorithms.

First, we had to decide on the model evaluation metric carefully, because using simply accuracy in case of imbalanced data set might be deceptive as with classifying the data as non-fraudulent will result in a stunning 99.83% accuracy without identifying a single anomaly. Instead, we decided to use here the F1-score as the leading metric which is the harmonic mean of the precision and recall metrics. Recall is measured as the fraction of cases classified as fraud by the model over the total fraudulent cases. Precision is measured as the fraction of true positive cases over the total cases classified as positive by the model. Precision is the complementary of the False Positive Rate, which also important to keep relatively low as possible because certain cost can be associated with it. From a business perspective, in case of credit card fraud detection, high recall is a score of primary importance, even if precision scores are mediocre, because the monetary cost of both performing secondary validation steps on all flagged card transactions (e.g. email/text, live over the phone), or declining *legitimate* card transactions flagged by such candidate models is usually considerably lower on average than the cost of allowing fraudulent transaction through (cost of low recall). Nevertheless, striving for high F1 scores are the gold standard in selecting between candidate models.

As a next step, we scaled those features that were not PCA components (transaction amount and time) and split our data set into train, validation and test set resulting in the same fractions of fraudulent transactions in each part:

	Normal transactions	Fraud transactions	Fraction of fraud
Train set	170,883	295	0.17%
Validation set	56,962	99	0.17%
Test set	56,962	98	0.17%

4.1 Unsupervised approaches

There is a wide range of unsupervised learning approaches we can employ in finding the outliers in our dataset. We have implemented the following methods:

Multivariate Gaussian distribution

The multivariate Gaussian distribution can be used for outlier detection in a simple but very efficient way, because it returns the probability that a new observations comes from specific given distribution. Given the train data set X where $x^i \in R^n$, its covariance matrix Σ, and the mean vector of the features μ, we can compute the probability of a new data point x that it comes from the above defined multivariate Gaussian distribution:

$$p(x) = \frac{1}{|\Sigma|^{1/2}(2\pi)^{n/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

The model uses only one parameter (ϵ), a threshold probability under which an observation will be considered as anomaly ($p(x) < \epsilon$). The Σ and μ parameters of the multivariate Gaussian distribution function was calculated only on normal transaction, however for selecting the best features and the threshold value we used the validation set.

Applying the Gaussian distribution in an univariate way (using only one feature at a time) was helpful to order the features by their importance. The following table shows the first few most predictive features, which was the basis of selecting the most efficient feature set in subsequent models as well:

Features	V14	V17	V12	V16	V3	V10	V11	V18
F1-score	0.600	0.557	0.514	0.442	0.278	0.257	0.244	0.239

The final multivariate Gaussian distribution uses only the first four (V14, V17, V12 and V16) features, because adding additional features to the model resulted in a decline in the model performance.

DBSCAN: Density-based spatial clustering of applications with noise

DBSCAN - as described in the paper [3] - is a clustering algorithm that formulates clusters based on the density of the data points. As opposed to the K-Means, this algorithm does not require to specify the number of clusters in advance (it is the output of the model) and also makes possible to create clusters of arbitrary shape. However, specifying the 'density' requires to set the following two parameters:

- *epsilon* parameter, that describes the maximum distance (using a distance metric by our choice) between two data points to be considered neighboring data points and assign them to the same cluster,
- *min_samples* parameter: the minimum number of neighbors within an *epsilon* distance as a requirement to assign a data point and its neighbors to a cluster. (Also referred as minPts parameter)

Data points that has *min_samples* number neighbors within an *epsilon* distance will be called as core-points, data points that are within an *epsilon* distance of a core-points, but doesn't have *min_samples* neighbors in *epsilon* distance is called as border points. Although DBSCAN was primarily designed for clustering, it also can be used in outlier detection, because data points that cannot be assigned to any clusters (there are no core points in an *epsilon* distance) are considered outliers.

We have trained our model using the same set of features as we used in the multivariate Gaussian distribution (V14, V17, V12 and V16). However finding the best set of parameters we had to tune the *epsilon* and *min_sample* parameteres together: as the *min_sample* gets larger it required to set the *epsilon* parameter larger as well. The final parameters that we have found efficient on the validation set was *min_samples* = 18, *epsilon* = 2.2. We should also see, that this model extremely sensitive to the density of the data set, which means that these parameters work only if we apply them to a very similar set of data. (Once we have tuned the data on two days of transactions, it will probably work on a similar scale of data.)

One-class SVM

The one-class Support Vector Method was described by Schölkopf [4], suggesting a model that learns the boundaries of the normal data points and separates all the data points from the feature space by maximizing the distance from this hyperplane. Similarly to the SVM we can apply non-linear kernel function to train the model and this can learn even non-linear boundaries of our normal transactions. The model should be trained on normal transactions, and the trained model can be applied on new data points. The function returns +1 for data points within the boundaries captured by the training data and returns -1 for data points out of these boundaries that can be considered as outliers.

The model uses *nu* parameter, which is similar to the C parameter in SVM model. This parameter controls the number of samples on the wrong side of the hyperplane but also gives a lower bound for the number of support vectors. We have trained one-class SVM using Gaussian RBF kernel method, in which case we also had to tune a smoothing parameter *gamma*. The final model was trained using only three features (V14, V17, V12).

The Local Outlier Factor

Local Outlier Factor [5] is a density based outlier detection techique, which computes the local density deviation of a given data point with respect to its neighbors. In the LOF algorithm, every data points gets a 'LOF score' assigned. Based on a pre-determined threshold T, this score is used for deciding whether a given data point is an outlier ($LOF_k(A) > T$) or a non-outlier ($LOF_k(A) > T$).

This algorithm calculates the following metrics:

1. the reachability distance from A' to A: $reachdist_k(A \leftarrow A') = max\{dist_k(A), dist(A, A')\}$
2. the local reachability distance of A: $lrd_k(A) = \frac{||N_k(A)||}{\sum reachdist(A' \leftarrow A)}$

The local reachability distance of A is the inverse of the average of the reachability distances of all the neighbors of A. If the local reachability distance value is high, it indicates that the neighbors of A tend to be far away, and therefore A is in a low density 'neighborhood'.

The Local Outlier Factor (LOF score) of A is the average of the local reachability distances of all A'-s in the k-distance neighborhood of A, divided by the local reachability distance of A:

$$LOF_k(A) = \frac{lrd_k(A')}{||N_k(A)||} \frac{1}{lrd_k(A)}$$

The threshold parameter was determined at the time of fitting the data from the identified 'contamination' level parameter, i.e. the (known) proportion of outliers in the training data set. Instead of domain expert input on the threshold, we assumed that the average 'contamination' level during the 2 day period that our dataset covers is indicative of the typical fraud rate, thus usable in determining the threshold at training time.

Isolation Forest

The Isolation Forest [6] algorithm is different from its other unsupervised cousins as it focuses on explicitly isolating outliers as opposed to profiling non-outlier datapoints. It isolates outliers by performing a recursive partitioning of the dataset:

1. Random selection of a set of features
2. Random selection of a split-point value(s) between the maximum and minimum values of the selected feature(s).

The splitting points are represented by a binary decision tree. The number of splittings required to isolate a datapoint (into a single element group, or leaf) a.k.a the path length of the leaf (single element group) is the focus of the algorithm: outlier datapoints are more 'different', therefore tend to be isolated quicker, i.e. they have a shorter path on the decision tree. The tree path length of a datapoint is averaged between a large number of randomly generated trees, and becomes the basis of an indicator of a point being outlier (decision function).

The key parameters turned out to be the maximum number of samples taken from the dataset in each iteration (max_samples = 350) and the maximum number of features based on which the split is performed (max_features= 5) After trying different values for it, the standard number of estimators (100) was found to be the value with which the algorithm performed best. We found that pre-selecting the important features and training the model on those *noticeably* improved the F1 score for the outlier detection from 0.32 to 0.75.

Summary of results

Table 1: **Unsupervised algorithms**

Algorithm	Recall	Precision	F1-score
Multivariate Gaussian distribution	0.847	0.703	0.769
DBSCAN	0.837	0.651	0.732
One-class SVM	0.827	0.711	0.764
Local Outlier Factor	0.888	0.159	0.270
Isolation Forest	0.796	0.703	0.746

Table 1 summarizes the results of the unsupervised algorithms. We can see, that these algorithms all performed well in identifying outliers in our credit card data set, however resulted in higher false positive rate (aka lower precision). While the 16% precision rate for the fraudulent transactions (label = 1) is quite low in case of LOF algorithm, the 89% recall is promising. This indicates, that 89% of fraudulent transactions will be flagged as fraudulent by this model which may be usable for certain business objectives.

4.2 Supervised learning approaches

We have implemented the following supervised learning algorithms:

1. Logistic Regression,
2. Random Forest
3. Artificial Neural Network (ANN)

In case of Logistic Regression model, we had to deal with the very imbalanced dataset to train the model efficiently. Our final approach to handle the imbalance dataset was the combination of the undersampling of the majority class (normal transactions) and synthesizing data points from the minority class (fraudulent transactions), creating a 4:1 more balanced ratio in the training set. This latter approach was described by Chawla, Bowyer, Hall & Kegelmeyer (2002) [2] in their paper "SMOTE: Synthetic Minority Over-sampling Tech-nique". The method takes a random instance from the minority class, randomly chooses one of its closest k-minority class neighbors, then creates a data point between these two instances in the feature space. This paper also points outs that a combination of the SMOTE oversampling of minority class and undersampling of majority class can achieve a better classifier performance in ROC space.

In case of Logistic Regression model we have applied both Lasso and Ridge regularization creating an Elastic-net model. The final model uses $\lambda = 200$ regularization parameter along with 0.4 weight for L1-norm (Lasso) and 0.6 weight for L2-norm (Ridge) regularization.

In case of Neural Network we have trained two different models using two different platforms (Sklearn and Tensorflow/Keras). Both models performed very similarly, however they had different layer structure and used different activation functions.

Both Logistic Regression and Random Forest algorithms allowed us to retrieve the most important features (V17, V14, V12, V10, V11, V4 just mentioning the most important), and we can see here some overlap with the features that turned to be important in case of multivariate Gaussian model.

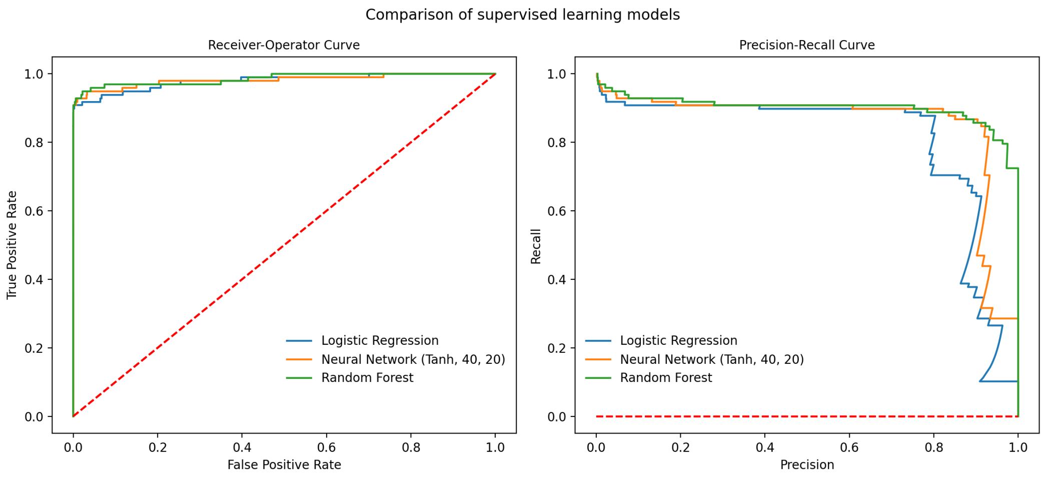
The following table summarizes the performances of supervised learning algorithms:

Table 2: **Supervised algorithms**

Algorithm	Recall	Precision	F1-score
Logistic Regression (Elastic-net)	0.867	0.802	0.833
Random Forest	0.837	0.943	0.886
Neural Network (Tahn, 40/20)	0.847	0.912	0.878
Neural Network (16/24/20/24/softmax))	0.87	0.86	0.86

We can see that these algorithms performed better than the unsupervised learning algorithms especially in precision rate which means these algorithms were able to find the fraudulent transactions with less false positive cases. Figure 4 shows the trade-off between the precision and recall, and based on this we can say that Neural Network and Random Forest algorithms perform slightly better than Logistic Regression.

Figure 4: Performance of supervised learning algorithms



5 Evaluation and final results

1. We can see that the **supervised learning algorithms performed usually better** than the unsupervised learning algorithms. The supervised learning algorithms usually resulted in just slightly better recall rate, but significantly higher precision rate compared to the unsupervised learning algorithms. The supervised algorithms were more efficient in reducing the false positive cases.

2. The most predictive and thus the **most important features** turned to be approximately the middle range (10^{th} to 17^{th}) out of the ordered 28 PCA components. It can be explained by the extremely imbalanced data set where the first few PCA components capture mainly the variance of the majority class (normal transactions), and the variance that is associated with fraudulent transactions will occur only in subsequent PCA components.

3. **Unsupervised learning algorithms usually performed better with a smaller set (3-6) of features** and required additional processing to find the best feature set and retrain the model with the selected features, however the supervised learning algorithms usually didn't require special feature selection as their algorithm involves this by tuning the hyperparameters.

4. Even the **unsupervised learning algorithms** use a set of hyperparameters, and the calibration of these parameters to find the very few outliers in the dataset **required a precise hyperparameter tuning**, expert considerations usually turned to be less effective. This implies that creating a well performing model using unsupervised learning algorithms requires **labelled data set** as well.

6 Identifying areas of further improvements

Although we have found algorithms that were able to find the majority of the fraudulent transaction with relatively low false positive cases, we can identify further areas of improvements. These areas might be an important part of answering real business question an thus put a similar model into a production.

1. Cost of false positive cases could be taken into consideration while setting the parameters of a model and/or creating and evaluating a metric. The amount of transactions was already included from which the cost of false negative cases can be calculated. Since the monetary cost of validating or potentially throwing away credit card transactions flagged by model is usually lower than the cost of allowing fraudulent transaction, combining it with the cost of false negative cases could allow us training and/or fine tuning our models such that reduces the overall cost associated with fraudulent transactions.

2. Adding customer identifiers would allow us to performing pre-clustering of users and then run our candidate machine learning algorithms on the individual clusters separately. We anticipate that this finer grained analysis would result in considerably better predictions because the outlier transactions could be identified with respect to the specific cluster rather than the whole data set. Straightforward pre-clustering can be done along with multiple dimensions, e.g. socio- economic status, customer interest profiles (e.g. learned from spending data), typical store purchases in various product/service categories, customer life-cycle events (e.g. learned from historical spending data), etc.

3. Adding location information – both the transaction location and the merchant's location. Another pre-filtering could be performed looking at impossible constellations of transaction locations (e.g. a transaction in San Jose, CA followed soon by another purchase in Paris, France.) Machine learning could be useful in identifying the outlier location constellations in the transaction stream, if customer identifiers would also be supplied. While some combinations of the PCA-d features are likely to preserve various levels of correlation with both customer ID and location data, the lack of them necessarily causes our analysis to result in lower accuracy than in case those would be included.

References

[1] https://www.kaggle.com/mlg-ulb/creditcardfraud

[2] Chawla, Bowyer, Hall & Kegelmeyer: SMOTE: Synthetic Minority Over-sampling Technique (2002)

[3] Ester, Kriegel, Sander, Xiaowei Xu: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise

[4] Scholkopf, Williamson, Smola, Shawe-Taylort, Platt: Support Vector Method for Novelty Detection

[5] https://scikit-learn.org/stable/auto_examples/neighbors/plot_lof_outlier_detection.html

[6] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html