Unsupervised vs. Supervised Learning on Credit Card Fraud Detection

Yan Chen
Computer Science Dept. of
San Jose State University
San Jose, CA, USA
yan.chen01@sjsu.edu

***Abstract*— Credit card fraud costs huge loss of money, and the traditional ruled-based method to detect fraud has a lot of limitations. Machine learning techniques are developed to increase accuracy. This paper explored both unsupervised (Local Outlier Factor) and supervised (Support Vector Machine) methods for credit card fraud detection by building models using existing data and comparing the result in accuracy, precision, recall, f1-score, and time to train the model.**

Keywords*— *fraud detection, Local Outlier Factor, Support Vector Machine, resampling

I. Introduction

As people use credit cards in various daily activities, fraud transactions will cost huge money loss for customers. The traditional way to detect fraud is rule-based, which depends on simple rules. For example, decline a transaction if the amount is above a certain value. Or if the cardholder who always uses the card in the US, it may be a fraud if the card is used in France. This method has a lot of shortcomings such as not capable of dealing with complex situations. Therefore, machine learning techniques are developed for more accurate fraud detection.

There are two board categories of machine learning algorithms for this problem. One is classification, or supervised, which involves classifying and labeling the data to be fraud or normal. Another one is anomaly detection, which is unsupervised. That is, the data are not labeled, and the outliers in the data should be fraud. In this project, both unsupervised learning algorithm, Local Outlier Factor (LOF), and supervised learning algorithm, Support Vector Machine (SVM) were performed for the fraud detection on a dataset from Kaggle [1]. And the result was compared in terms of accuracy, precision, recall, f1-score, and time to train.

This project used scikit-learn library from Python and the code was written and run by Jupyter notebook. The rest of this paper is organized as the following: section II demonstrates the data processing, including scaling and resampling (balancing the data); section III introduces the implementation details for both algorithms; section IV compares the training results followed by a conclusion that summarizes the result and future work. Fig. 1 shows an overview of the project.

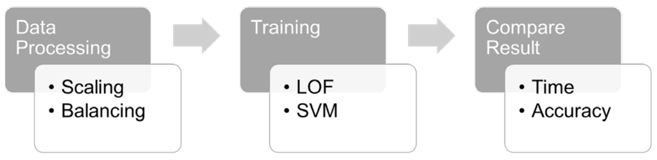

Fig. 1. Project overview: project started with data processing, then train and compared the models.

II. Data Processing

The data from Kaggle [1] are the transactions in September 2013 by European cardholders, which is in .csv format. The data are labeled so that 1 is for fraud and 0 is for normal. It does not contain any null values but contains duplicates. In this project, the duplicates are removed since they do not provide any additional information but would cost more space and time. The original data contain 284,807 rows and is reduced to 283,726 rows after removing the duplicates.

There are 31 columns include a column for the class (label). Since normally +1 and -1 are used for binary classification in machine learning, the labels were changed so that +1 representing the fraud transactions and -1 representing the normal transactions. Next sub-sections discussed scaling and resampling the data in detail.

A. Scaling

The remaining 30 columns are features for the data. Except for 'Time' and 'Amount', other 28 features are hidden by transforming to principle components due to confidentiality. A principal component (PC) is a normalized linear combination of the original features in a data set. PCs are uncorrelated to each other but captures the variance of data, as shown in Fig. 2. And Principal Component Analysis (PCA) is often used to reduce the dimension without losing important variables. Therefore, no need to do anything with the 28 hidden features since they are already PCA transformed.

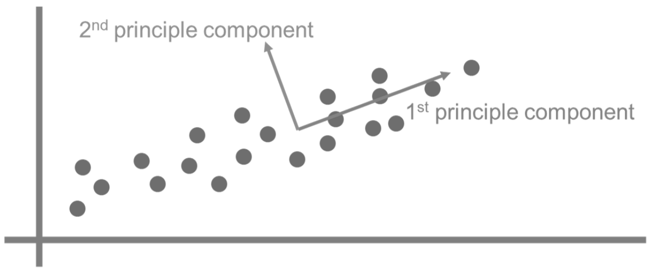

Fig. 2. Principal component example for 2-dimentional data.

'Time' is the seconds between each transaction and the first transaction. For example, the first transaction has time 0, and the transaction will have time 60 if it happens 1 minute after the first one. And 'Amount' is the amount spend in each transaction. The range for 'Time' is from 0 to 172,792 while the range for 'Amount' is from 0 to 25,691; the two ranges differs a lot. Therefore, the data under 'Time' and 'Amount' need to be scaled so that none of the feature will weigh in a lot more in the distance calculations used to build the models.

The standard scaler scales the data based on the mean and standard deviation. However, the data have outliers which can affect the result of computing the mean and standard deviation. To solve this problem, the project used a robust scaler, which uses percentiles to scale the data, so the scaling will not be influenced by outliners [2]. After scaling, the time has a range from -0.995 to 1.035 and the amount has a range from -0.306 to 356.962.

B. Resampling

Not surprisingly, the data only contain 0.17% of fraud transactions. This imbalance problem not only occurs in this dataset, but also occurs for many other classification problems such as malware detection. It is not hard to understand since a behavior is only called normal if it happens almost all the time. This kind of unbalancing data will cause the model to be biased towards the majority class, since the model only learned about the normal data. Since it is naturally hard to collect more data in the minority class, resampling is often used to balance the data.

Traditionally, there are two ways for resampling. Either under sampling the data, which means decreasing the number of data in the majority class by randomly remove the data from that class; or over sampling the data, which means increasing the number of data in the minority class by randomly duplicate data in minority class [3]. However, neither of these two ways can apply in this situation since the gap between the two classes is too large. A lot of data will be lost if use under sampling – the number of data in majority will need to decrease from over 200 thousand to less than 500. And use over sampling will cause overfitting since the minority data will need to be duplicated by 500 times in average.

This project combined both under sampling and over sampling techniques. First, random under sampling [4] the data so the ratio of number of minority data and number of majority data increased from 0.0017 to 0.0026. This is for decreasing the gap for the minority class to catch up, and for canceling out the new data that will be generated when oversampling so that the number of data remains roughly the same.

Then, for oversampling, this project used Synthetic Minority Over-sampling Technique (SMOTE) [5] combined with Edited Nearest Neighbor (ENN) [6] [7] . Instead of randomly reduplicating the data in minority class, SMOTE first find k-nearest neighbors (KNN) of the minority data. In this project, k = 3 was used. Then the synthetic new data will be the points along those vectors between each data point in minority class and its nearest neighbor as shown in Fig. 3. In this way, the new data are similar to the real data without duplicating the data in the minority class.

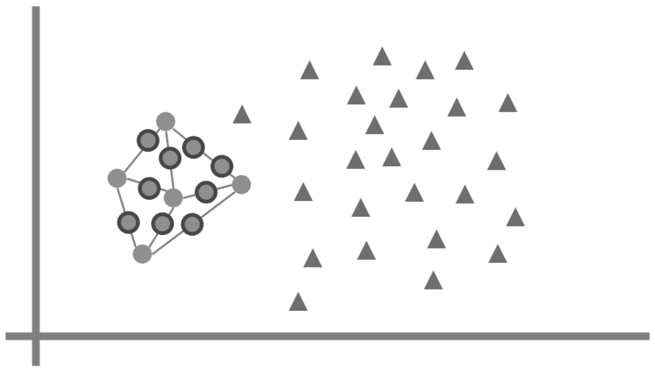
Fig. 3. Illustration for SMOTE: round dots are minority while triangles are majority. Round dots with outline are the synthetic new data.

ENN is used for data cleaning which is also based on KNN. First, find the KNN of the majority data. Again, the project used k = 3. As shown in Fig. 4, a point will be removed if it has a neighbor with more data of the other class (minority class). Therefore, ENN removes all borderline or noisy data to form a smoother decision surface [8].

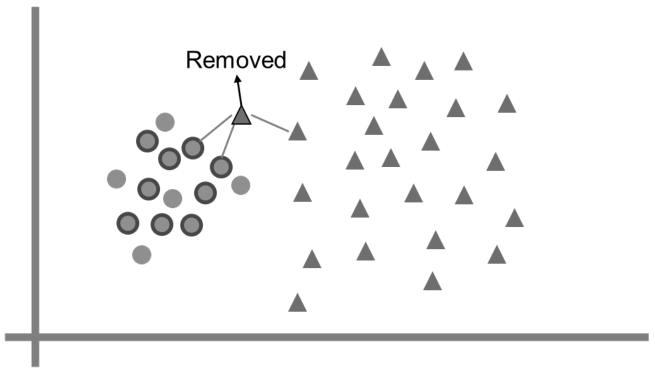

Fig. 4. Illustration for ENN: round dots are minority while triangles are majority. The triangle with outline is removed since there are more data of minority class in its 3-nn.

Note that after resampling, the two classes do not have the same size of number in this project. Instead, the final ratio between minority class and majority class is around 1:2. This is because the unbalance itself can also be considered as an information in the data, which should not be removed totally. As mentioned above, the total size of data remains about the same. Result was compared to the situation that final ratio = 1 (size of data increased).

1) Resampling and Cross validation

There is a common mistake of resampling when using cross validation. The wrong way is resampling the data before splitting data to n-folds for the cross validation. In this way, especially when using over-sampling, the test data will include the synthetic data. But the test should be done only on the real data; testing on made-up data does not make sense. Instead, the data should be split to n-folds first, then only resample the folds of data for training. The portion of the data for testing should remain the same; do not change anything. Fig. 5 demonstrates the correct way for cross validation when resampling is needed.

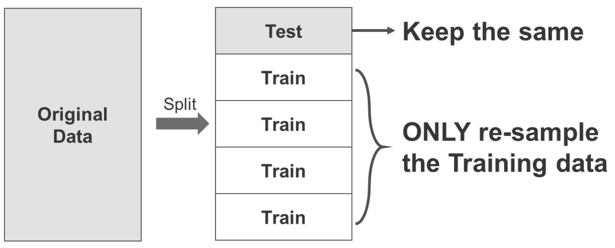

Fig. 5. Correct way for cross validation when resampling is needed: split the data first, then only resample the training data.

This project processed a 5-fold cross validation for evaluating the models. Therefore, in this project, the data was first divided to 5 folds using scikit-learn [9] for cross validation, then only the training data were resampled through the resampling techniques discussed above.

III. Implementation

This section briefly explained the models and the parameters used in this project. The models were implemented using scikit-learn library [10] [11] from Python.

A. Local Outlier Factor (LOF)

LOF [10] is an unsupervised anomaly detection method that based on the local density of data. Locality is given by nearest neighbors and density is calculated by their distance. The data points that have lower density then their neighbors are considered as outliers, as shown in Fig. 6. In this project, the outliers are the fraud transactions.

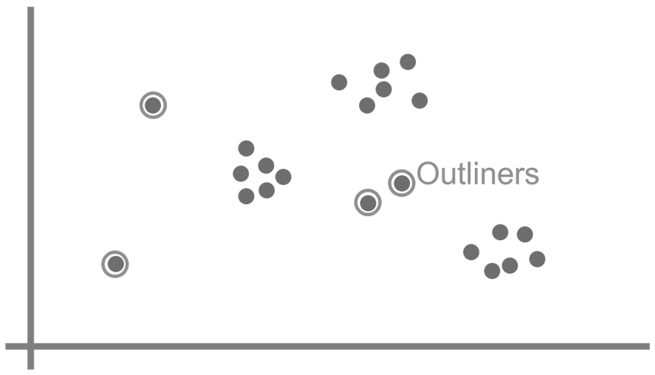

Fig. 6. Illustration for LOF: circled dots are the outliners.

The main parameter for this algorithm is the number of neighbors. Ideally, it should be set between the minimum number of data needed to form a cluster and the maximum number of potential outliers that are close-by. However, this information is rarely available. In this project, 20 is used since it appears to work well in general [10]. Theoretically, LOF does not affected by the unbalanced data since it only focusses on local density.

B. Support Vector Machine (SVM)

SVM is a supervised method used for binary classification. It can be applied in this project since the data are labeled. To separate the two classes of data, SVM tries to find a hyperplane that maximize the margin, which is the minimum distance from the hyperplane to any element of the dataset [12], as shown in Fig. 7.

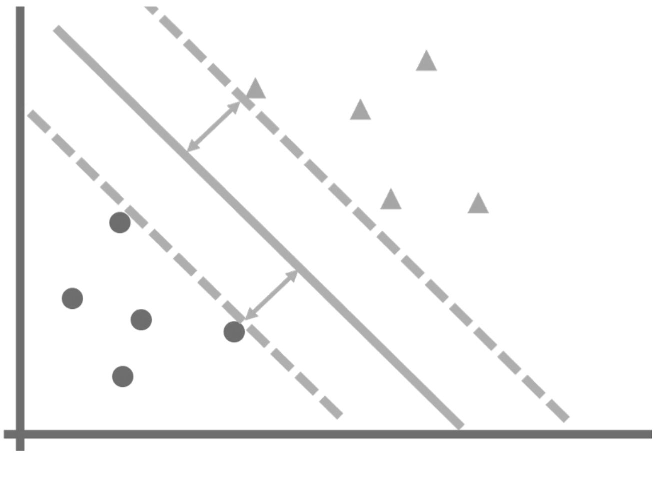

Fig. 7. Illustration for SVM in for 2-dimensional data: the solid line is the separating hyperplane and the dotted lines are the margin.

For data that is not linear separable or high-dimensional data, a kernel function depends on dot-products is used to transform the data to so that they are separable, as shown in Fig. 8. Therefore, SVM is effective for high-dimensional data since more available space means a higher chance of finding a separating hyperplane [12]. Since the data have 30 features, which means it is a high-dimensional dataset, SVM is expected to work well in this project both in terms of accuracy and efficiency.

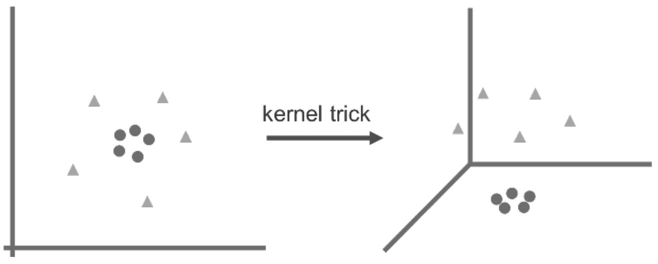

Fig. 8. SVM for non-linear-separable data: kernel trick (function) is used to transform to higher dimension so that the data become separable.

The main parameters for SVM are parameter C and kernel function. The parameter C is used for all kernel function, which determines the simplicity of hyperplane. C should be positive, and a smaller C makes a larger margin, resulting in a simpler hyperplane but at the cost of accuracy, while a larger C allows a smaller margin for classifying more data correctly [13]. In this project, a common value C = 1.0 was used.

Two different kernel functions were used and compared in this project, including Radial Basis Function (RBF), and polynomial kernel. Equation (1) is the RBF, while equation (2) is for polynomial, where x and x' are two different feature vectors in the data and $\langle x, x' \rangle$ is the dot-product. [13].

$$\exp\left(-\gamma \|x - x'\|^2\right) \tag{1}$$

$$\left(\gamma \langle x, x' \rangle + r\right)^d \tag{2}$$

The common parameter that RBF and polynomial functions shared is gamma (γ), which defines the influence of a single training data. The smaller the gamma is, the further other data are affected [13]. The value used in this project was multiplicative inverse of number of features (30) multiply by the variance of the training data.

There are two other parameters for polynomial kernel function, d for degree and r is a constant which often sets to 0 (so did this project). The degree of polynomial kernel has the similar effect as gamma. It controls the flexibility of the SVM model, the lower the degree, the less flexible decision boundary [14]. In this project, d is set to 3.

IV. Experiments and Result

This project tested on LOF and SVM with different kernel functions. Different resampling combinations, including random under sampling + SMOTE + ENN with final ratio = 0.5 or with final ratio = 1, and random under sampling + SMOTE with final ratio = 0.5, were also tested and compared. To control the variables, all models were trained using same data, both unsampled and resampled, for each iteration in 5-fold cross validation.

A. Measures

For binary classification, there are four possible outcomes as summarized in table I. In this project, fraud transactions were considered as positive. Thus, the false negative rate is the rate that fraud transactions are misclassified as normal, which ideally should be to 0.

TABLE I. FOUR POSSIBLE OUTCOMES FOR BINARY CLASSIFICATION

Actual	**Predict**	
	Positive (fraud)	***Negative (normal)***
Positive (fraud)	True Positive	False Negative
Negative (normal)	False Positive	True Negative

The models are evaluated by accuracy, precision, recall and f1-score. The accuracy is percentage of correct prediction, which may not have all the information for unbalanced dataset as demonstrated in Fig 9. To obtain information such as false positive rate, precision, recall and f1-score are often used. Precision is the ratio of correct positive prediction to all positive prediction so that a higher precision means a lower false positive rate. Recall is the percentage of correct positive prediction over all actual positive cases so that a higher recall means a lower false negative rate. And the F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account and it is usually more useful than accuracy when we an unbalanced data.

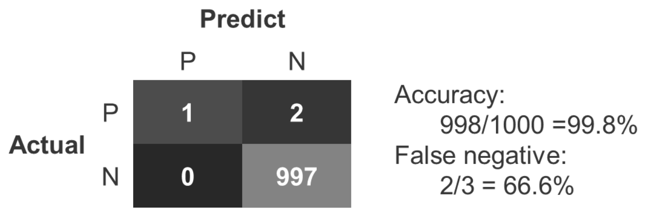

Fig. 9. A sample confusion matrix: high accuracy but also high false negative rate – only 1/3 fraud are correctly classified.

B. Results

Fig. 10 shows the average execution time for resampling the data of each fold, excluding the time for under sampling. SMOTE was way faster than ENN while changing ratio did not affect the time much.

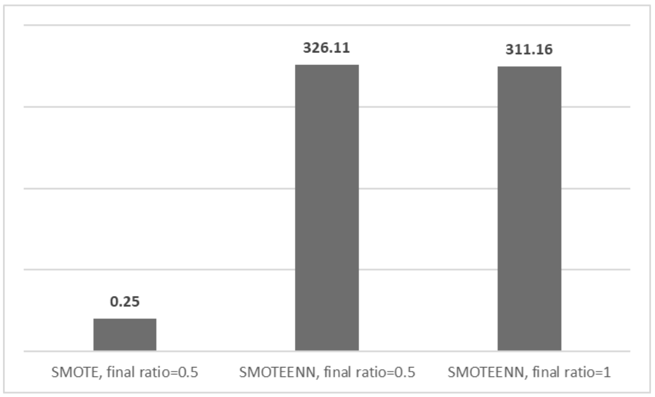

Fig. 10. Average resampling time for each fold in seconds (excluding the time for random under sampling.

For the average time needed to train the data, SVM models were much faster than LOF. This is because SVM is efficient for high-dimensional data, while LOF is not. The training time was also related to the data size as expected. The size of data decreased when the final ratio after resampling was 0.5, so did the time to train the models. And when final ratio was 1, which means more training data, the time increased. Result are shown in Fig. 11.

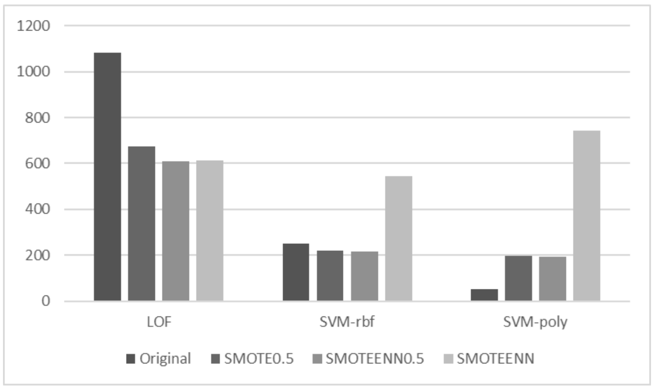

Fig. 11. Average time to train the model for each fold.

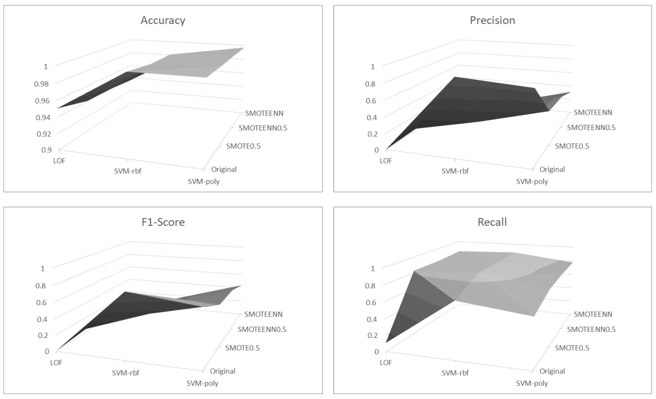

Fig. 12. All four scores for different resampling combination and different models.

Fig. 12 shows all the scores using different resampling combination and different models. The x-axis are different models, y-axis are different resampling combinations, and z-axis are the corresponding scores. Overall, SVM had higher scores compared to LOF. Result did not change much with different kernel functions. The recall score increased after resampling for all models. But the precision score decreased significantly resulting in lower f1-scores after resampling. Below is the detailed analysis.

1) Effect of resampling

For the accuracy, although did not change much, but the accuracy decreased after resampling as shown in Fig. 13. This may cause by decreased precision scores. As shown in Fig. 14, resampling increased the recall score for all models, especially for LOF. Without resampling, LOF had low precision and recall scores. The recall score increased from 10% to 80% while precision score did not improve much. And surprisingly, for SVM, the precision decreased significantly after resampling, resulting in a lower f1-score as shown in Fig. 15. Cleaning the borderline points using ENN had a little effect on the scores, while more synthetic data (when final ratio = 1) decreased the scores.

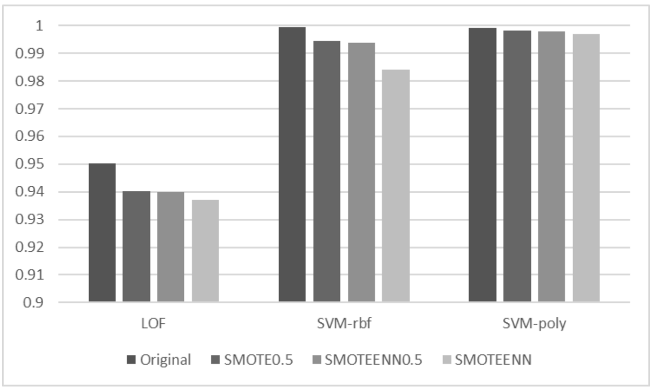

Fig. 13. Accuracy score.

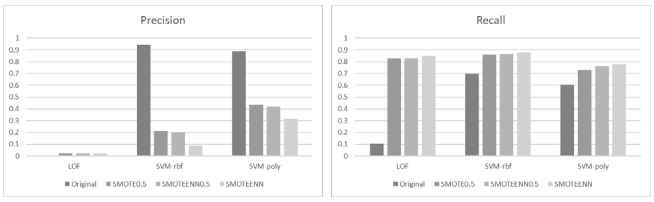

Fig. 14. Precision vs. Recall

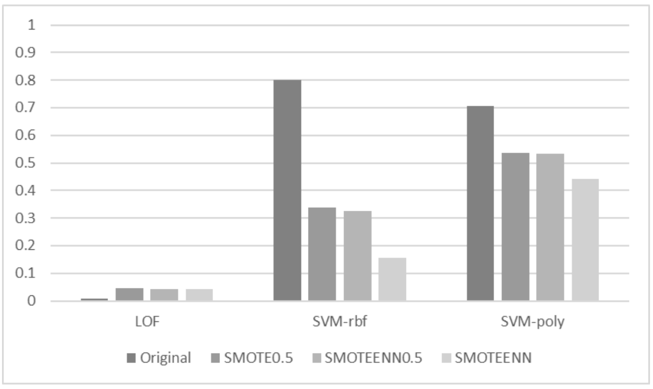

Fig. 15. F1-score

A lower precision means a higher false positive rate. That is, the model misclassified normal as fraud a lot. So, the results implied the synthetic data from SMOTE added many fraud data that are similar to the normal data. However, resampling also gave higher recall scores, which means the model has a better ability to classify the fraud from the normal. Thus, the synthetic data did give the model more data to learn about the fraud. For fraud detection, a lower false negative is more important than a lower false positive. Therefore, resampling is needed for this kind of problems.

2) *Comparing models*

SVM had an overall better performance compared to LOF. LOF is based on local density while SVM is trying to find a separating hyperplane. SVM worked better than LOF even without resampling indicated that in this case, the fraud data are clustered and separable from the normal data rather than have low density. Therefore, the techniques that are not density-based may be more suitable for credit card fraud detection.

V. Conclusion

This project experimented LOF and SVM for credit card fraud detection using an existing dataset on Kaggle. Resampling techniques such as SMOTE and ENN were used to balance the original data. The result showed that SVM trained the data faster and got higher accuracy as well as other scores compared to LOF. Different kernel function did not affect the result much.

Resampling using SMOTEENN increased the recall but decreased the precision much more, resulting in decreasing in F1 score. While using SMOTE only and combining with ENN did not change the results much, all scores decreased when the number of data in the minority class is increased to the same as in the majority class. The above observation implied although the models can learn more about fraud through synthetic data, most of synthetic data were similar to the data in majority class causing a high false positive rate even after using ENN to clean the borderline data.

Therefore, future research is needed to test on more resampling techniques, especially for cleaning undesired synthetic data, and other machine learning models using anomaly detection or classification to find an optimum solution. A technique that is not density based may be better for this kind of problem. Once an optimum model is found, programs or applications can be developed to detect credit card fraud.

VI. References

[1] "Credit Card Fraud Detection," Kaggle, [Online]. Available: https://www.kaggle.com/mlg-ulb/creditcardfraud.

[2] "Compare the effect of different scalers on data with outliers," scikit-learn, [Online]. Available: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html.

[3] A. Fernández, S. García, M. Galar, R. Prati, B. Krawczyk and F. Herrera, "Chapter 5 Data Level Preprocessing Methods," in *Learning from Imbalanced Data Sets*, Springer, 2018, p. 80.

[4] "imblearn.under_sampling.RandomUnderSampler," imbalanced-learn, [Online]. Available: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.under_sampling.RandomUnderSampler.html.

[5] R. Blagus and L. Lusa, "SMOTE for high-dimensional class-imbalanced data," *BMC Bioinformatics,* vol. 14, 2013.

[6] "imblearn.under_sampling.EditedNearestNeighbours," imbalanced-learn, [Online]. Available: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.under_sampling.EditedNearestNeighbours.html.

[7] "imblearn.combine.SMOTEENN," imbalanced-learn, [Online]. Available: https://imbalanced-

learn.readthedocs.io/en/stable/generated/imblearn.combine.SMOTEENN.html.

[8] M. Beckmann, N. F. F. Ebecken and B. S. L. P. d. Lima, "A KNN undersampling approach for data balancing," *Journal of Intelligent Learning Systems and Applications,* vol. 7, no. 4, p. 104, 2015.

[9] "sklearn.model_selection.KFold," scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html.

[10] "Outlier detection with Local Outlier Factor (LOF)," scikit-learn, [Online]. Available: https://scikit-learn.org/stable/auto_examples/neighbors/plot_lof_outlier_detection.html.

[11] "sklearn.svm.SVC," scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[12] M. Stamp, "A Reassuring Introduction to Support Vector Machines," in *Introduction to machine learning with applications in information security*, Boca Raton, CRC Press, 2018, pp. 95-125.

[13] "Support Vector Machines," scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/svm.html.

[14] A. Ben-Hur and J. Weston, "A User's Guide to Support Vector Machines," in *Data Mining Techniques for the Life Sciences*, Humana Press, 2010, pp. 223-239.