This is the project report submitted for Term Project CIS 732- 2022 SPRING

Credit Card Fraud Detection

Krish Weragalaarachchi
Kansas State University

Abstract:

With the 2020 global pandemic, there is a major shift in online purchases using credit/debit cards. Keeping pace with that trend is an unfortunate increase in credit card fraud in which thieves use other people's credit card information on card-not-present (CNP) space to make purchases. Supervised and unsupervised Machine Learning models can be used to effectively identify fraud and stop them before happening.

Introduction:

According to the FBI, credit card fraud is "the unauthorized use of a credit or debit card or similar payment tool to fraudulently obtain money or property." There were nearly 400,000 frauds reported to Federal Trade Commission in 2020 and it is a 44% increase from 2019. Credit card fraud loses billions of dollars from consumers, merchants, and financial intuitions. The key to minimizing this kind of fraud is to detect it early and act immediately to stop it. It is getting harder to detect fraud as fraudsters use techniques that look more legit to make fraudulent transactions and they keep finding new ways. Therefore, financial institutions need better models to identify credit card fraud.

Fraud Detection with Machine Learning is possible due to the ability of Machine Learning algorithms to learn from historical fraud patterns and recognize them in future transactions. Machine Learning algorithms are more effective and efficient compared to humans and they also can identify more complex fraudulent patterns that humans cannot easily detect.

Supervised Machine Learning models can be used to successfully detect fraud and, in this project, I compared 7 binary classification techniques which are recommended for fraud detection including bagging and boosting classifiers. Usually, the vast majority of the transactions are legit and a very small minority will be fraudulent. Hence, transaction data is usually severely imbalanced. This leads predictive models to overfit on majority class while ignoring the minority class. There are multiple methods have been suggested to treat the class imbalance and I tested 4 of those techniques with the 2013 European cardholders' dataset which is publicly available on Kaggle.

Related Work:

There are so many examples and articles available online on credit card fraud detection. Many of them use this dataset. There is not much transaction data available for public use as transaction data contains confidential information this dataset is one of the few available transaction datasets. That must be the reason for using this dataset by many researchers.

Data Set:

The dataset used in this project is originally released as a part of a research collaboration of Worldline and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection. It contains credit card transactions from European cardholders in 2013. The dataset is highly imbalanced, with 492 fraudulent transactions out of the 284,807 total transactions. As the original features have been transformed for confidentiality using PCA, the dataset contains 28 PCA components

and two features that have not been transformed. They are ‘Amount’ and ‘Time’. Amount refers to the transaction amount, and Time refers to the seconds elapsed between any transaction in the data and the first transaction. This data is publicly available on Kaggle platform.

Methodology:

The data is checked for null, missing, or nan values and duplicate entries. There were 1081 duplicate entries and they were removed before further processing. The ‘Time’ feature refers to the time taken between the transaction and the first transaction in the dataset and does not represent useful information. Thus, the ‘Time’ feature was converted to ‘DateTime’, and the hour component was extracted in an attempt to identify if the hour the transaction happened is correlated with the possibility of a transaction being fraudulent. Here ‘Hour’ is a categorical value representing an hour of the day. This way, all transactions that happened in the same timeframe can be identified even though we do not have a way to identify what exactly is the hour of the day it is happened (Figure 01)

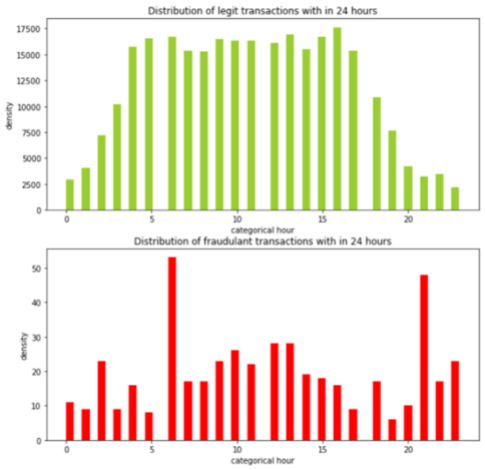

Figure 01: Distributions of legit and fraudulent transactions for categorical ‘Hour’. Fraudulent transactions have been increased during categorical hours 6 and 21.

To study feature importance, the feature correlation matrix, the density distribution of two classes for all features, and estimated mutual information between each feature and the target variable were calculated.

V8, V13, V15, V20, V22, V23, V24, V25, V26, and V28 have similar distribution shapes for fraud distribution and legit distribution. Therefore, we can conclude that they are not contributing much to the classifying power of the model. The distributions for fraud and legit classes of V15 V13 and V22 are depicted in Figure 02.

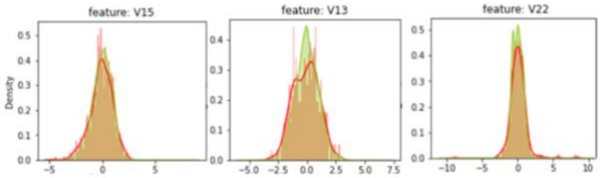

Figure 02: class distributions for V15, V13, and V22

The features which have mutual information between each feature and target variable less than 0.001 are depicted in Table 01.

Feature	mutual_infomation score
V20	0.000904
V23	0.000471
V24	0.000364
V25	0.000251
V26	0.000239
V22	0.000126
V13	0.000085
V15	0.000023

Table 01: features with mutual information score less than 0.001

I used all the features for modeling as there is no domain knowledge to remove these features as they are PCA components. Removing some of the features to see whether there is an improvement in the classifying power of the selected best model is an option but it is not the scope of this project.

Then the ‘Amount’ and ‘Hour’ attributes were standardized by removing the mean and scaling to unit variance. The dataset is 144MB in size and a sample (0.25) maintaining the same class imbalance ratio (fraud: legit = 1:600) was selected for model building as the whole data set is too big for me to use with my limited computer power and resources. Then the sample dataset was split into 80-20 train-test split.

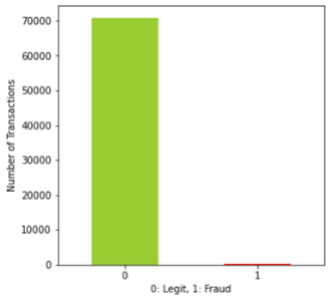

Figure 03: Distribution of legit and fraud in sample maintaining the same imbalance ratio of the initial dataset.

The dataset is severely imbalanced. It is a property of this domain in such that usually, the number of fraud transactions that happens during a certain period is less compared to the number of real transactions that happens during the same period. However, a severe class imbalance is a challenging problem in model building as standard classifiers tend to overwhelm by majority class and ignore the minority class. As we are interested in identifying the fraud which is the minority class, we have to apply special techniques to treat the imbalance before using the data for model building. There are number of different methods to treat class imbalance and I have used "random oversampling", "Synthetic Minority Over Sampling Technique" (SMOTE), a combination of "SMOTE" and "undersampling" and "Borderline Synthetic Minority Oversampling technique which uses Simple Vector Machines" (SMOTESVM) to treat the class imbalance of training set before using it for model building.

Random oversampling

Random oversampling involves randomly selecting examples from the minority class, with replacements, and adding them to the training dataset.

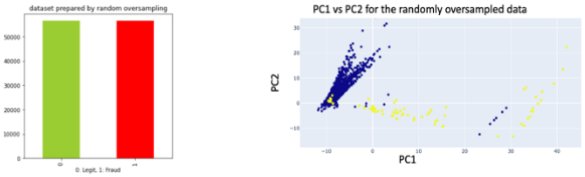

Figure 04: a) class distribution of training set after random oversampling to treat the class imbalance b) PC1 vs PC2 of the random oversampled data.

Synthetic Minority Over Sampling Technique (SMOTE)

This synthesizes new points based on the K-Nearest Neighbor algorithm. Specifically, a random example from the minority class is first chosen. Then, k nearest neighbors for that example are found (default k=5). One neighbor from k is randomly selected and a synthetic example is created at a randomly selected point between the two examples in feature space.

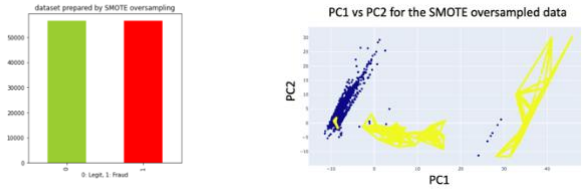

Figure 05: a) class distribution of training set after SMOTE oversampling to treat the class imbalance b) PC1 vs PC2 of the SMOTE oversampled data.

SMOTE + Undersampling

First minority class was oversampled to have 50 percent of the number of examples of the majority class, then the number of examples in the majority class was reduced using random undersampling to have a 1:2 ratio between minority class and majority class (fraud: legit)

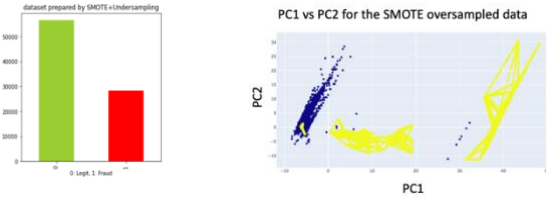

Figure 06: a) class distribution of training set after SMOTE + random under-sampling to treat the class imbalance b) PC1 vs PC2 of the SMOTE + random undersampled data.

Borderline SMOTE SVM

SVM is used to locate the decision boundary defined by the support vectors and examples in the minority class that are close to the support vectors become the focus for generating synthetic examples. This selects regions, where there are fewer examples of the minority class and, tries to extrapolate toward the class boundary.

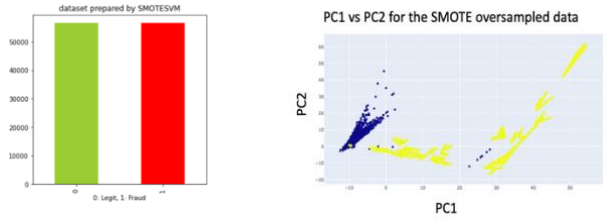

Figure 07: a) class distribution of training set after SMOTESVM oversampling to treat the class imbalance b) PC1 vs PC2 of the SMOTESVM oversampled data.

The training datasets which treated with the above 4 techniques for class imbalance were tested with 7 different binary classification algorithms to find the best model for the data. They are as follows;

1. Logistic Regression
2. K-Nearest Neighbor (with optimal k =3)
3. Naïve Bayes
4. Simple Vector Machines (SVM)
5. Decision Tree
6. Random Forest (Bagging Ensemble)
7. XGBoost (Boosting Ensemble)

The logistic regression model is selected as the baseline model.

The testing set was used for model evaluation and 35 combinations (7 classifiers with imbalanced dataset and 4 different imbalanced treated datasets) were evaluated by Precision, Recall, F1 score, and area under the Precision-Recall curve.

The best classifier-imbalance removing technique combination I found was XGBoost with SMOTESVM treated dataset. The performances of the XGBoost classifier and the baseline were evaluated using RepeatedStratifiedKFold, repeating 2 times a 5-fold stratified cross-validation using a different randomization of the data in each repetition. Stratified means that each fold will contain the same mixture of examples by class, that is about 99.8 percent to 0.2 percent legit and fraudulent transactions respectively. Repeated means that the evaluation process will be performed multiple times to help avoid fluke results and better capture the variance of the chosen model. I have used 2 repeats. This gives each model a sample of 2 scores for 5-fold cross-validation. A single model will be fit and evaluated 10 times and the mean and standard deviation of these runs will be reported. The scores were then compared by the paired student t-test using the paired_ttest_5x2cv() function implemented in the MLxtend library for statistical significance of the observed difference in the performance scores of the two classifiers.

There are different ways to perform parameter search. The Exhaustive Gridsearch and Successive Halving GridSearch are two of them. Exhaustive Gridsearch exhaustively searches over specified parameter values for an estimator and Successive Halving GridSearch starts evaluating all the candidates with a small amount of data and iteratively selects the best candidates for the estimator, using more and more data. Exhaustive Gridsearch and Successive Halving GridSearch were compared for performance efficiency using 3 parameters to find the best parameters for the XGBoost ensemble classifier which uses SMOTESVM treated dataset. The parameters I chose are 0, 0.25, and 1 for gamma which specifies the minimum loss reduction required to make a split, 3, 4, 5, and 6 for maximum depth of the tree (max_depth), and 1 and 100 for min_child_weight which defines the minimum sum of weights of all observations required in a child. Altogether, 24 models were tested. The performance was evaluated in each model using RepeatedStratifiedKFold repeating 2 times a 5-fold stratified cross-validation using a different randomization of the data in each repetition. The performance of models was evaluated using average precision.

Evaluation:

All of the initial 7 models (35 combinations; 7 classifiers with imbalanced dataset and 4 different imbalanced treated datasets) were evaluated using precision, recall, F1-score, and area under the precision-recall curve. The area under the precision-recall curve is obtained using the average_precision_score() function from the scikitlearn library. It summarizes a precision-recall curve as the weighted mean of precisions achieved at each threshold, with the

increase in recall from the previous threshold used as the weight:

$$\mathrm{AP} = \sum_n (R_n - R_{n-1}) P_n$$

Where P_n and R_n are the precision and recall at the n^{th} threshold. The logistic regression classifier with imbalanced dataset was selected as the baseline. For the hypothesis test, the models were evaluated using the F1-score and for the Gridsearch comparison, the models were evaluated on average precision.

Results:

Precision, Recall, F1-Score, and average precision values for all of the tested 35 model-data combinations are depicted in Table 02. Among the 35 model-data combinations I tested, the XGBoost ensemble classifier when tested with SMOTESVM treated dataset showed the most optimal precision and recall combination which resulted in the highest F1-score and highest average precision score compared to the baseline values.

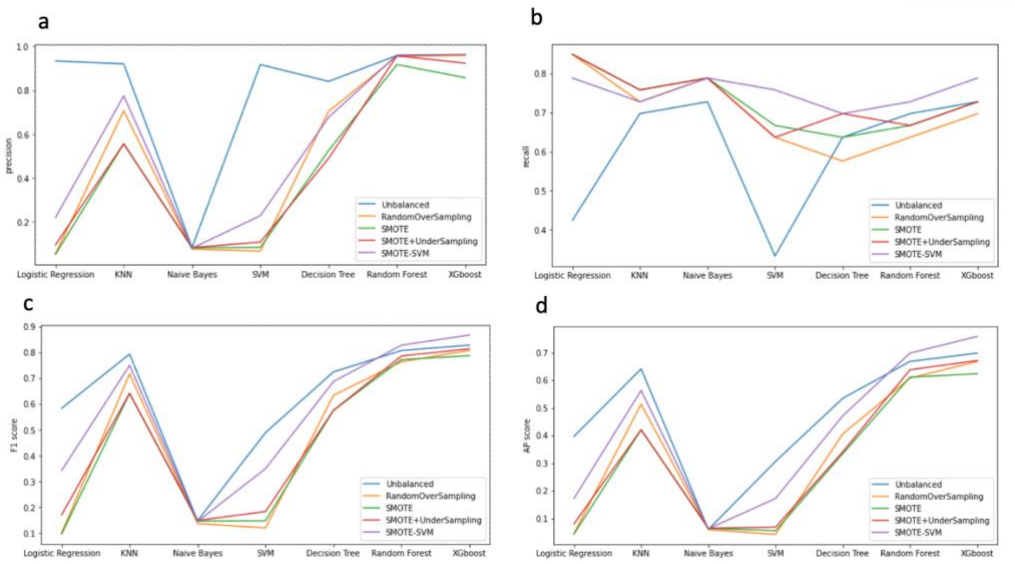

Figure 08: a) precision b) recall c) F1-score d) average precision for 35 model-data combinations

	Imbalance Technique	Logistic Regression	KNN	Naïve Bayes	SVM	Decision Tree	Random Forest	XGBoost
Precision	imbalanced	0.9333	0.92	0.0822	0.9167	0.8519	0.9583	0.96
	RandomOversampling	0.0534	0.7059	0.0749	0.0662	0.6774	0.9565	0.9583
	SMOTE	0.0511	0.5556	0.0807	0.083	0.5	0.9167	0.8571
	SMOTE+Undersampling	0.0946	0.5556	0.0823	0.1071	0.4894	0.9565	0.9231
	SMOTESVM	0.2185	0.7742	0.0795	0.2273	0.6765	0.96	0.963
Recall	imbalanced	0.4242	0.697	0.7273	0.3333	0.697	0.697	0.7273
	RandomOversampling	0.8485	0.7273	0.7879	0.6364	0.6364	0.6667	0.697
	SMOTE	0.8485	0.7576	0.7879	0.6667	0.6061	0.6667	0.7273
	SMOTE+Undersampling	0.8485	0.7576	0.7879	0.6364	0.697	0.6667	0.7273
	SMOTESVM	0.7879	0.7273	0.7879	0.7576	0.697	0.7273	0.7879
F1-Score	imbalanced	0.5833	0.7931	0.1477	0.4889	0.7667	0.807	0.8276
	RandomOversampling	0.1005	0.7164	0.1368	0.12	0.6562	0.7857	0.807
	SMOTE	0.0964	0.641	0.1465	0.1477	0.5479	0.7719	0.7869
	SMOTE+Undersampling	0.1702	0.641	0.149	0.1834	0.575	0.7857	0.8136
	SMOTESVM	0.3421	0.75	0.1444	0.3497	0.6866	0.8276	0.8667
Average Precision	imbalanced	0.3973	0.6419	0.0604	0.3071	0.5944	0.6686	0.6988
	RandomOversampling	0.0457	0.514	0.0595	0.043	0.4319	0.6385	0.6686
	SMOTE	0.0437	0.4214	0.0641	0.0561	0.3039	0.6119	0.624
	SMOTE+Undersampling	0.0806	0.4214	0.0653	0.069	0.3418	0.6385	0.672
	SMOTESVM	0.1726	0.5637	0.0631	0.1727	0.4722	0.6988	0.7592

Table 02: precision, recall, F1-score, average precision for 35 model-data combinations. The baseline model is logistic regression

The performance of the XGBoost classifier and the logistic regression classifier (for SMOTESVM data) was compared using repeated stratified k-fold cross-validation with 5 folds and 2 repeats. Figure 09 depicts the box and whisker plot summarizing the distribution of F1-scores. The mean F1-score for the logistic regression classifier is 0.445 and that for the XGBoost classifier is 0.834. The P-value obtained from paired student t-test is 0.036 < (p = 0.05). This suggests that the observed difference between the XGBoost classifier and logistic regression classifier is statistically significant.

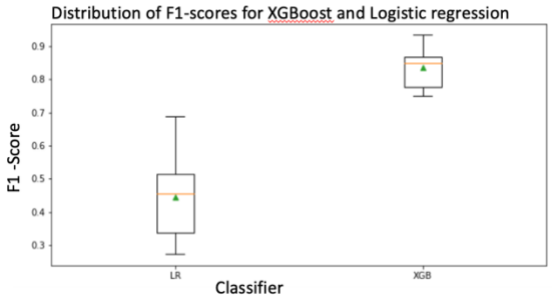

Figure 09: Box and whisker plot summarizing the distribution of F1-scores

The loss function for the XGBoost classifier is depicted in figure 10.

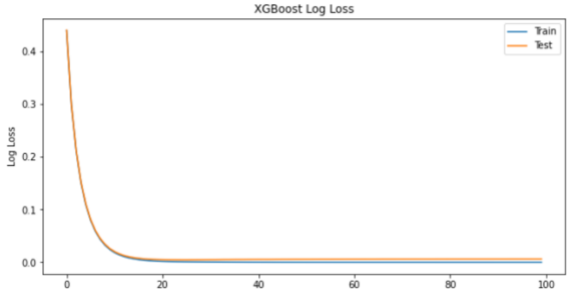

Figure 10: Loss function for XGBoost

The best parameters obtained for Exhaustive Gridsearch are gamma = 1, max_depth = 3, max_child_weight = 1 and that for Successive Halving Gridsearch are gamma = 0, max_depth = 3 and min_child_weight = 1. The search time is significantly lower for Successive Halving Gridsearch. It is 14.61 minutes for Successive Halving Gridsearch and 62.32 minutes for Exhaustive Gridsearch. The best score was evaluated based on average precision and Exhaustive Gridsearch showed a higher score of 0.7973 compared to that of 0.7898 of Successive Halving Gridsearch. The best parameters max_depth and min_child_weight obtained from Exhaustive Gridsearch and Halving Gridsearch are the same except for gamma(Table 03).

	Exhaustive GridSearch	Halving GridSearch
Best parameters	gamma: 0.25, max_depth: 4, min_child_weight:1	gamma: 0, max_depth: 4, min_child_weight: 1
Best average precision score	0.7973	0.7898
Total search time (minutes)	62.32	14.61

Table 03: comparison between Exhaustive and Successive Halving Gridsearch

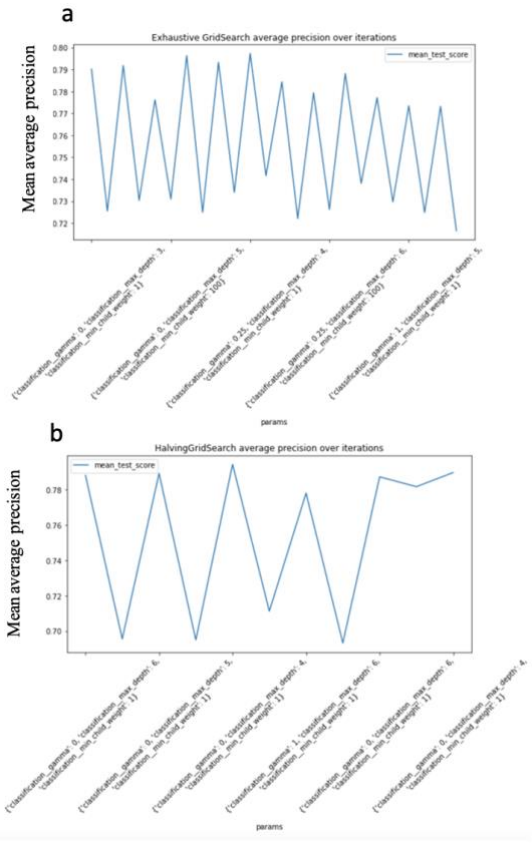

Figure 11: mean average precision of parameter grid space for a) Executive GridSearch b) Successive Halving Gridsearch

Conclusion:

From all the model-data combinations that I tested, the XGBoost ensemble classifier when tested with SMOTESVM treated data is more effective in predicting fraud in the 2013 European credit card dataset over the baseline

model; the logistic regression classifier (p = 0.036). Naïve Bayes algorithm and SVM algorithms performed poorly as they failed to maintain an optimal blend of precision and recall.

Successive Halving Gridsearch is computationally and timely more efficient than Exhaustive Grid search for Hyperparameter tuning. It should be noted that Successive Halving Gridsearch is still in experimental stage but as can be seen from figure 11-b it reached the optimal solution. Exhaustive Gridsearch suffers from curse of dimensionality and it does not guarantee the best solution but often aliasing over the best configuration (figure 11-a)

In all, this project is challenging and rewarding. Handling of imbalance data is a crucial part of many disciplines these days and getting to know available techniques to treat imbalance data is very important to any researcher. I used 4 different methods to treat the class imbalance and there are many others. Extending the imbalance treating techniques would be a good learning experience. Moreover, I used the SMOTE oversampling technique with default k(k=5). Finding the optimal k for SMOTE's inheriting K-Nearest Neighbor algorithm could have resulted in different precision, recall, F1-score, and average precision for classifiers when used with SMOTE oversampling technique. The effectiveness of SMOTE oversampling technique might have been better than SMOTESVM if the optimal k was used. This is something we can implement and research in the future.

I only used a portion of the dataset due to limited computer resources. More generalized scores would have been obtained if the entire dataset was used. In addition to that, I had to limit the parameter search to 3 parameters due to extended computational time. First, I tried with 6 parameters (576 combinations). The parameters I tried are; 0,0.5,1 and 5 for gamma, 3,4 5 and 6 for max_depth, 1 and 100 for min_child_weight, 0.6, 0.8, and 0.1 for colsample_bytree and 0.1 and 0.01 for learning_rate. However, the search did not finish even after 48 hours and the search had to limited to first 3 parameter and 24 combinations. Extending the parameter grid in a wide range would have helped me to find a better more generalized model with a high optimal blend of precision and recall.

Finally, the features that have low mutual information scores and same class distributions could have been removed to see whether the performance score increases.

References:

1. N.V. Chawla, Data mining for imbalanced datasets: An overview, In Data mining and knowledge discovery handbook, pp. 875-886, 2009.
2. Soleymani, Ali, Stop using SMOTE to treat class imbalance, take this intuitive approach instead, 2021, medium.com
3. S. Venkata Suryanarayana, G. N. Balaji, G. Venkateswara Rao, Machine Learning Approaches for Credit Card Fraud Detection, 2018 doi: 10.14419/ijet.v7i2.9356
4. Megasari Gusandra Saragih, Jacky Chin, Rianti Setyawasih, Phong Thanh Nguyen, K. Shankar , Machine Learning Methods for Analysis Fraud Credit Card Transaction, 2019
5. Jason Brownlee, SMOTE for Imbalanced Classification with Python, Machine Learning Mastery, 2020
6. Mitchell, T. M. (1997). Machine Learning. New York: McGraw-Hill. ISBN: 978-0-07-042807-2
7. Murphy, K. P. (2012). Machine learning: a probabilistic perspective. Cambridge, MA: MIT Press.
8. Han, H., Wang, WY., Mao, BH. (2005). Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In: Huang, DS., Zhang, XP., Huang, GB. (eds) Advances in Intelligent Computing. ICIC 2005. Lecture Notes in Computer Science, vol 3644. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11538059_91
9. Pedregosa et al., Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011.
10. T. G. Dietterich, "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms," in Neural Computation, vol. 10, no. 7, pp. 1895-1923, 1 Oct. 1998, doi: 10.1162/089976698300017197.
11. Raschka, (2018). MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack. Journal of Open Source Software, 3(24), 638, https://doi.org/10.21105/joss.00638