Credit Card Fraud Detection Machine Learning Project
Group 3

Introduction

Credit card fraud is the most common identity theft. According to Credit Card Fraud 2021 Annual Report, fraud has become more common since the pandemic. Although the amount per transaction is generally small, the aggregated amount reached multibillion dollars during the first year of the pandemic. Reports increased 44 percent, according to the Federal Trade Commission (2022).

While these statistics are U.S.-centric, fraud undoubtedly is a global problem. Based on historical transaction data and machine learning, we aim to build a fraud detection model that may better prevent fraud in the future.

Data Understanding and Preparation

We obtain Kaggle's Credit Card Fraud Detection data. The dataset contains transactions made by European cardholders in two days during September 2013. There are 492 frauds out of 284807 transactions, accounting for only 0.17% of all transactions.

It only contains numeric input variables resulting from a Principal Component Analysis (PCA) transformation, an unsupervised learning algorithm often used to reduce dimensionality and preserve privacy (Brownlee, 2020). Because of confidentiality issues, the provider cannot give the original features and background information but only labels them from V1 to V28.

Specifically, the attributes in the dataset include the following:

- V1-V28: the principal components and the results of PCA Dimensionality reduction to protect user identities and sensitive parts.

- Time and Amount: the only features that have not been transformed with PCA. The time contains the seconds elapsed between each transaction and the first transaction in the dataset. Amount describes the transaction amount. They both belong to continuous data.

- Class: the feature takes value one if the transaction is fraudulent and 0 if the transaction is legitimate.

Before sending the data to the model, we explore the amount and time further because they have not been transformed with PCA. Therefore, we can interpret the information more easily.

To investigate the amount, we look at the basic statistics and compare the top five most frequent legitimate transactions with the top five fraudulent transactions. We find that the fraudulent transactions have a higher mean and standard deviation but an overall lower amount (Figure 1).

Interestingly, the small amount mirrors the recent American cases mentioned in the introduction. This observation is important because it proves that even though the dataset only covers the transactions in two days and was created almost ten years ago, the data still appears to be relevant and thus meaningful.

As for time, the fraudulent transactions spread more evenly than the legitimate transactions, indicating that frauds occur more often than valid ones during nighttime in the European time zone (Figure 2).

While we do not spot any missing value in the dataset, the class imbalance is a severe issue in this case. We visualize this imbalance as shown in Figure 3. In the next section, we discuss the different ways we adopt to solve this issue.

We run a correlation using the unbalanced dataset to understand better why it is essential to deal with class imbalance. Figure 4 shows that there are almost no identifiable correlated relationships. Although the result is not surprising, this step helps us better understand the poor predictive performance of imbalanced class data and the importance of having an even distribution of classes to obtain more meaningful results.

Modeling

As mentioned, we cannot observe any notable pattern using the imbalanced data. And we discovered more than one way to deal with this, modeling with the unbalanced dataset directly or resample to get a balanced dataset first. As a group that shares a business background yet relatively limited programming knowledge, we figure that trying different algorithms to get a good grasp on the programming field serves the group's best interests and the learning outcome, rather than applying one algorithm and coming up with the utmost performance.

Therefore, we embrace a wide variety of methods. For the cohesion of this report, we first discuss what we do to deal with the unbalanced class directly. Namely, we adopt weighted logistic regression and XGBoost. Then, we introduce several resampling methods, including random undersampling, random oversampling, SMOTE oversampling, and a combination of the random undersampling and SMOTE oversampling. After each resampling method, we have subsections documenting the different modeling techniques. With each modeling technique, we also try different parameters.

Method 1: Weighted Logistic Regression

We find that using class weights in logistic regression is common when working with imbalanced distribution (Yadav, 2020). Class weighting adjusts the model's cost function so that misclassifying fraudulent cases can be more heavily penalized.

First, we do not apply a class weight (i.e., lr = LogisticRegression(max_iter=1000)) in order to test the model. As shown in Table 1, we get a recall of 0.58 and an F1 score of 0.69. That is, the model can only detect 58% of the frauds. We choose the recall of fraudulent cases and the F1 score to be indicators of a good model for the following reasons.

The recall value is important in our case because we want to know the true frauds out of any transactions that have been predicted as positive. Thus, the reason to use recall is straightforward. A high recall indicates that we can successfully identify the frauds. In addition, the F1 score is the harmonic mean of precision and recall. More importantly, based on our understanding, F1 score is useful when there are imbalanced classes.

In our case, because the samples of fraudulent transactions are so few, one can get a high accuracy by simply predicting that the transactions are legitimate all the time. This would be not very useful and arguably harmful if we let go of all the fraudulent cases, albeit the number and amount of them are small. However, with an F1 score, if the recall for the frauds is zero, the F1 score would be zero as well. This is why the F1 score comes in handy for our case.

Then, we apply a balanced weight (i.e., lr_balanced = LogisticRegression(class_weight = 'balanced', max_iter=1000)). It should 'automatically adjust weights inversely proportional to class frequencies in the input data.' (D'Angela, 2021). In this scenario, the recall value is slightly improved (i.e., 0.88). However, the model only has 0.09 for the F1 score,

resulting from a more serious Type 1 error than when we do not assign any class weight. Here, we may argue that simply applying a ‘balanced’ class weight may not be the optimal solution.

We try to find a more suitable weight. We manually put 1, 5, 100, and 500 as weights for class 1, keeping class 0’s weight unchanged at 1. We document interesting changes in recall and F1 scores when we adjust the weights.

Specifically, the recall and F1 scores are 0.56 and 0.66 when the weight is 1 for class 1, and the values are 0.83 and 0.82 when the weight is 5. Increasing the weight to 100, we get a high recall (i.e., 0.90), meaning that the model correctly gets 90% frauds. However, it is interesting to see that the F1 score plummets to 0.38. Since F1 presents a balance between precision and recall, the poor score tells us that the precision of the model is getting lower. We hypothesize that F1 would get lower if we further increase the weight for class 1. The result confirms our hypothesis. While the recall hits 0.96 when the weight is 500, the F1 score only reaches 0.12. Here, we reflect that this happens because class 1 is the minority class, and if we increase its weight, we are more likely to get false positives.

Hyperparameter Tuning

In addition, we also explore hyperparameter optimization because one of the purposes is to maximize the model's performance. Specifically, we tried different C values, whereas a high C value gives a high weight to the training set. Hence, it is reasonable to have a wide range of C values from low to high. Here are the values that we adopt: 0.001, 0.01, 0.1, 1, 10, 100, 1000. As shown in Table 2, the best C is 1.

Method 2: Extreme Gradient Boosting (XGBoost)

XGBoot is a relatively advanced and efficient supervised learning algorithm. Some articles suggest that it is ten times faster than the nominal Gradient Boosting (Goyal, 2020). We choose this because we would like to explore more about ensemble techniques, which should achieve better performance than any single model and be more robust due to the wisdom of crowds.

The results are not surprising yet impressive. They are better than those in logistic regressions, achieving a 0.96 recall and a 0.87 F1 score and getting 157 frauds correctly from 164 fraudulent cases. This confirms our prior understanding that ensemble learnings

are effective algorithms. However, like any other tool, there are certain trade-offs. We observe that XGBoost is evidently more time-consuming to run than logistic regression. Hence, if speed is the requirement, perhaps XGBoost is the less desirable choice.

Next, we apply different resampling methods to achieve data balancing, including random undersampling, random oversampling, SMOTE oversampling, and the combination of SMOTE oversampling and random undersampling. These choices are logical because they share different strengths and shortcomings. We see this project as an opportunity to explore the above techniques and better grasp each method.

Method 3: Random Undersampling

We believe undersampling is a relatively simple and effective approach to address an imbalanced dataset problem. In brief, undersampling takes random samples from the majority class equal to the number of samples in the minority class. We first split the training test for the entire dataset and then apply the undersampling to the training set. Here, we use the RandomUnderSampler to select the normal cases, making the number of cases the same as that of frauds; each class has 293 samples in the training data. Although this method suffers from several disadvantages, it ultimately reduces the bias associated with class imbalance (Bilogur, 2018).

We understand that undersampling can discard potentially valuable data, and the data we select could be biased. Consequently, the result may be inaccurate. With these disadvantages in mind, later, we adopt oversampling, where no information from the original training set is lost.

Modeling techniques

Logistic Regression: We adopt this technique for several reasons. First, it is easy to implement and efficient to train, predicting the probability of the target variable. The second reason is related to the first one; the target variable is dichotomous (binary) in our case, being either fraudulent or a legitimate transaction. This touches on the essence of logistic regression, which is a statistical method to predict a binary outcome.

As demonstrated in Table 1, with undersampling and default parameter values, logistic regression returns a high recall value (i.e., 0.87) in the testing dataset. With a high recall value, naturally, the False Positive (FP) value is high (i.e., 5552), meaning that many transactions are predicted as fraudulent when these transactions are actually legitimate.

We infer that this may lead to a high risk of overestimating the fraudulent incidence and lowering the operating efficiency, imposing a higher cost on operations.

In addition, we test different parameter values for the logistic regression model. Table 1 documents the trials and their performance. It is important to note why we change our performance metrics to accuracy and recall rather than keep using the F1 score. The primary reason is that the F1 score, as mentioned, is more meaningful when we need to evaluate the model performance on an unbalanced dataset. Additionally, with a balanced dataset, we use accuracy to grasp the general model performance. Also, we keep the recall because it is more suitable than precision for security matters in fraud detection.

We recall that a balanced class weight leads to poor performance when we have an unbalanced dataset; hence, we wonder how assigning a balanced class weight to the undersampled dataset might change the performance. Intriguingly, the accuracy stays at 0.95, and the recall value increases to 0.90. The FP value drops to 5350, and the FN value drops from 26 to 20. The result contrasts with when we have a balanced class weight for the unbalanced dataset. Since undersampling is the only new factor, we believe it is safe to assume that it is the undersampling process improving the performance in this case.

Decision Tree: As mentioned in the class, the popularity of this method leads the pack in data mining competition. It is an upside-down tree-like, supervised learning model for predicting the value of a target. Hence, we are curious to see the result. Preliminarily, without any pruning or change in the threshold, we get a significantly lower recall value (i.e., 0.01) and a significantly lower FP (i.e., 28) than the logistic regression. Nonetheless, the False Negative (FN) spikes to 12157 (versus 20 in logistic regression). As mentioned, in the credit card fraud detection, we believe that a high FN could be more consequential than a high FP. In other words, letting actual frauds getaway is more severe than catching false alarms. Intuitively, the high FN can fail to detect fraud, and it is logical to assume that the customer may lose trust in the credit card company that fails to do so.

Hence, we believe that we should test different parameters. Recalling that a simpler tree may have better generalization performance, we go for pruning by setting different values for max_depth. We observe an inverse relationship between the max depth and recall and accuracy value. In other words, the lower the max depth, the higher the values (Table 1). However, the best performance from the decision tree still returns a relatively poorer

result than other methods. Although the accuracy can hit as high as 0.97, the recall remains low, potentially leading to negative business implications. Therefore, we conclude that the decision tree is not the best choice.

Method 4: Random Oversampling

Random oversampling is to duplicate the samples from the minority class to achieve a balance. Like random undersampling, this is a very naive strategy. We use the term 'naive' because they do not make any assumptions about data. However, unlike undersampling, oversampling keeps all the data in the training set. Of course, oversampling has its limitations. Since it simply makes copies of the samples from the minority class, this resampling method increases the probability of overfitting, increasing the generalization error. Additionally, we are also aware that increasing samples also increases computational costs. Nevertheless, random oversampling is a robust model (Ling and Li, 1998), and we think it is worth to be tested.

Similar to the approach in the undersampling process, we split the training and testing dataset first and then import RandomOverSampler from imblearn.over_samplin, resampling for the training set. As a result, we get 170591 for each class.

Modeling techniques

Logistic Regression: for comparison purposes, we believe it is reasonable and necessary to apply the same modeling techniques after each different resampling method. Again, we start from the most basic model (i.e., lr_oversample = LogisticRegression()). As the dataset is balanced, there is no logical reason to change the measurement for performance; thus, we keep recall and accuracy as the indicators for a good model. With an oversampled training dataset, the default logistic regression performance is better than the undersampled dataset. As the best performing model so far, it returns a recall of 0.90 and an accuracy of 0.97. It is noteworthy that the model gets 180 fraudulent transactions correctly out of 199 frauds from the testing dataset.

Further, we apply different parameters; however, as shown in Table 1, none of them outperforms the default model. In general, the accuracy scores turn out to be better than those in the undersampled dataset.

Decision Tree: Intriguingly, the performance of the decision tree improves significantly. The default model gives us a recall of 0.79 and an accuracy of 1.00. In addition, the AUC

is 0.87. This result is rather surprising to the best of our knowledge because random oversampling simply duplicates the minority samples. Arguably, this only improves the quantity instead of the quality of the data, while the latter is also imperative for machine learning (Gudivada, 2017). While we conclude that random oversampling improves the decision tree model, in this case, we do not find any evidence indicating that solely boosting the data quantity can guarantee a better performance of the decision tree. Additionally, we test different values of max depth. We observe that pruning does not necessarily increase accuracy as the tree does not learn enough parameters.

Random Forest: This is a supervised learning algorithm and an ensemble learning method. It comprises randomly created decision trees and is efficient on a large dataset. Therefore, we choose to adopt this technique after resampling with oversampling rather than undersampling because the former significantly increases the size of the dataset.

When training and testing, we notice that the method seems to be slower than logistic regression and decision tree, and our research confirms this observation (Donges, 2022). Digging further, we find that ensembles of decision trees are fast to train but slow to make predictions. However, the random forest classifier returns the best recall and accuracy scores, 0.93 and 1, respectively. This mirrors what we have done in the class: an ensemble of models is generally more accurate than any of its individual members.

In short, if run-time performance is not a concern, the random forest classifier is an excellent predictive modeling tool for credit fraud detection.

Method 5: SMOTE Oversampling

Another popular oversampling method is Synthetic Minority Oversampling Technique (SMOTE). The last random oversampling's repetition of the original samples does not increase the variety. In contrast, oversampling using SMOTE creates artificial training samples and increases the variety. Here, we use SMOTE to rectify the class imbalance because this is a widely used approach and because we can create synthetic data points that are different from the original ones. We understand that SMOTE may oversample uninformative or noisy samples.

Meanwhile, we notice that SMOTE is slower than random under and oversampling. However, it is worth testing it out for several reasons. First, as mentioned, the method is popular. More importantly, we are curious and expect to see different results from the

random oversampling. By importing SMOTE and fitting training data, we again have 170591 for each class.

Modeling techniques

Logistic Regression: With SMOTE, the default logistic regression's recall is higher than undersampling but lower than oversampling, and accuracy is the highest among the three. Also, changing parameters does not give a better result. Unexpectedly, SMOTE does not return one of the best results. We trace back to the original paper of SMOTE, seeking the reason (Chawla, 2002). It turns out the 'combination of [the] method of oversampling the minority class and undersampling the majority class' should achieve better performance (ibid). Therefore, we combine the two in the next section.

Decision Tree: We again observe a high precision and low recall using the default method and changing the parameters. Due to limited length, we document the results in Table 1. Even though the precision is high, this can be detrimental to credit card fraud detection because it misses many fraudulent cases. Therefore, the decision tree model is not desirable.

Random Forest: Since this classifier achieves the best result in the previous random oversampling, we expect a similar, if not better, result by applying SMOTE. However, the recall is only 0.88; the score is 0.7 lower than that with random oversampling. We find this part somewhat out of expectation because our preliminary research on SMOTE tells us that the algorithm is famous, popular, and advantageous (e.g., Brownlee, 2020; Manchev, 2021). This leads us to anticipate Random Forest, the best model we have, and SMOTE should outperform.

Method 6: Combination of Undersampling and Oversampling

According to the original SMOTE paper (Chawla et al., 2002), the combination of under and oversampling might achieve a better result. The idea is to oversample the minority class and undersample the majority class. To implement this in Python, we utilize the sampling_strategy attribute. As suggested by the paper, we use SMOTE oversampling and random undersampling. Due to limited length, we only test out the default logistic regression. The results do not appear to be desirable (Table 1).

However, it is reasonable to assume that the combination method is less time-consuming than SMOTE as it reduces a part of the data. We may try different parameters to achieve a more desirable outcome in the future.

Discussion and Conclusion

Table 1 shows all the methods, techniques, and parameters we try. The combination of Random Forest and random oversampling gives us the best outcome, with recall equal to 0.93 and accuracy being 1.00, getting 93% of the frauds correctly.

However, we recognize several shortcomings of the report. First, the dataset only covers two days of the transactions in Europe. Naturally, this causes time and geographic constraints. While, as mentioned in the introduction, the patterns still apply to other countries or regions such as the U.S., we should bear in mind that the different periods and locations could impact the model performance, especially when credit fraud detection is a global problem for a long time. Second, it is important to note that many methods and combinations are still to try. In our report, we only adopt the common supervised learning methods. In the future, we may also test supervised learning.

Overall, this project is a valuable learning opportunity allowing us to have a deeper, more comprehensive understanding of different algorithms by applying them to a real-world business problem. Although we go through most of the techniques theoretically and practically in class, the process is still full of surprises when we have hands-on experience. Also, we find it interesting that the most popular method (i.e., Decision Tree) is not necessarily the best one. Arguably, one never fits all, and finding the most suitable algorithm for the specific business case is important.

Reference

Bilogur, A. (2018, April 9). Undersampling and oversampling imbalanced data. Kaggle. Retrieved May 15, 2022, from https://www.kaggle.com/code/residentmario/undersampling-and-oversampling-imbalanced-data/notebook

Brownlee, J. (2020, August 18). Principal component analysis for dimensionality reduction in python. Machine Learning Mastery. Retrieved May 15, 2022, from https://machinelearningmastery.com/principal-components-analysis-for-dimensionality-reduction-in-python/

Brownlee, J. (2021, March 16). Smote for imbalanced classification with python. Machine Learning Mastery. Retrieved May 15, 2022, from https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16, 321-357.

D'Angela, A. (2021, February 4). Why weight? the importance of training on balanced datasets. Medium. Retrieved May 15, 2022, from https://towardsdatascience.com/why-weight-the-importance-of-training-on-balanced-datasets-f1e54688e7df

Donges, N. (2022). Random Forest Algorithm: A complete guide. Built In. Retrieved May 15, 2022, from https://builtin.com/data-science/random-forest-algorithm

Federal Trade Commission . (2022, May 5). Company Identity Theft Report Types. Free data visualization software. Retrieved May 15, 2022, from https://public.tableau.com/app/profile/federal.trade.commission/viz/IdentityTheftReports/TheftTypesOverTime

Goyal, S. (2020, April 28). *Boosting performance with XGBoost*. Medium. Retrieved May 15, 2022, from https://towardsdatascience.com/boosting-performance-with-xgboost-b4a8deadede7

Gudivada, V., Apon, A., & Ding, J. (2017). Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations. International Journal on Advances in Software, 10(1), 1-20.

Manchev, N. (2021). Synthetic minority oversampling (smote) in ML: Techniques & Examples. Synthetic Minority Oversampling (SMOTE) in ML: Techniques & Examples. Retrieved May 15, 2022, from https://blog.dominodatalab.com/smote-oversampling-technique

Security.org Team. (2022, March 29). Credit Card Fraud 2021 annual report: Prevalence, awareness, and prevention. Security.org. Retrieved May 15, 2022, from https://www.security.org/digital-safety/credit-card-fraud-report/

Yadav, D. (2020, April 14). Weighted Logistic Regression for Imbalanced Dataset. Medium. Retrieved May 15, 2022, from https://towardsdatascience.com/weighted-logistic-regression-for-imbalanced-dataset-9a5cd88e68b

Ling, C. X., & Li, C. (1998, August). Data mining for direct marketing: Problems and solutions. In Kdd (Vol. 98, pp. 73-79).

Appendices

Figure 1. Description of normal vs. fraudulent transactions amount

normal.describe()	frauds.describe()
count 284315.000000	count 492.000000
mean 88.291022	mean 122.211321
std 250.105092	std 256.683288
min 0.000000	min 0.000000
25% 5.650000	25% 1.000000
50% 22.000000	50% 9.250000
75% 77.050000	75% 105.890000
max 25691.160000	max 2125.870000
Name: Amount, dtype: float64	Name: Amount, dtype: float64

Figure 2. Description of normal vs. fraudulent transactions time

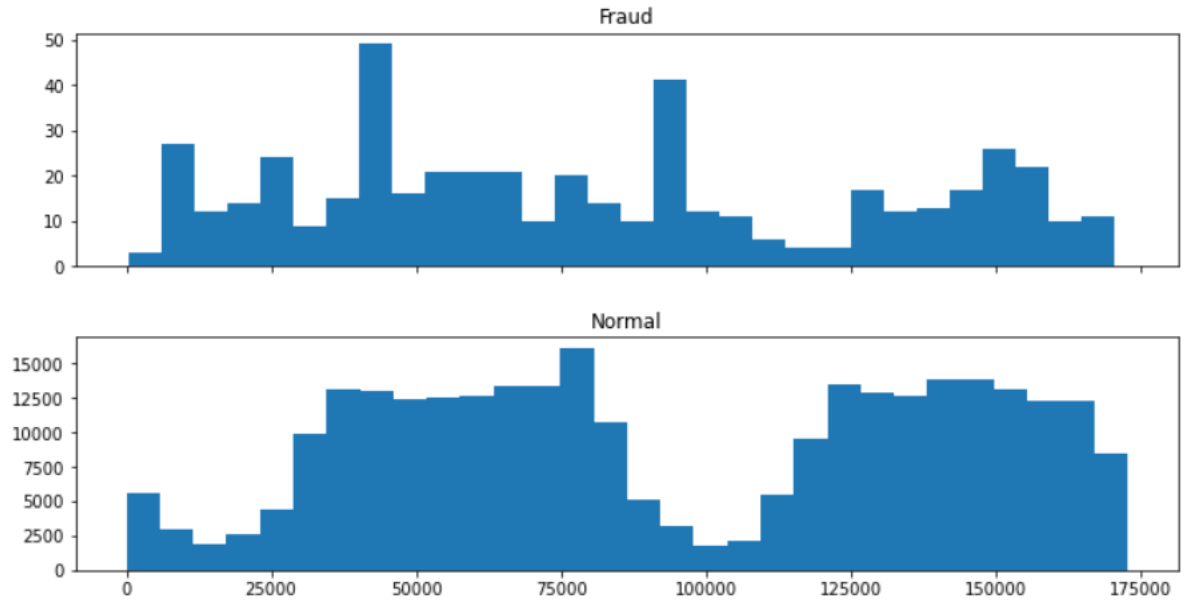

Figure 3. The frequency of normal vs. fraudulent transactions

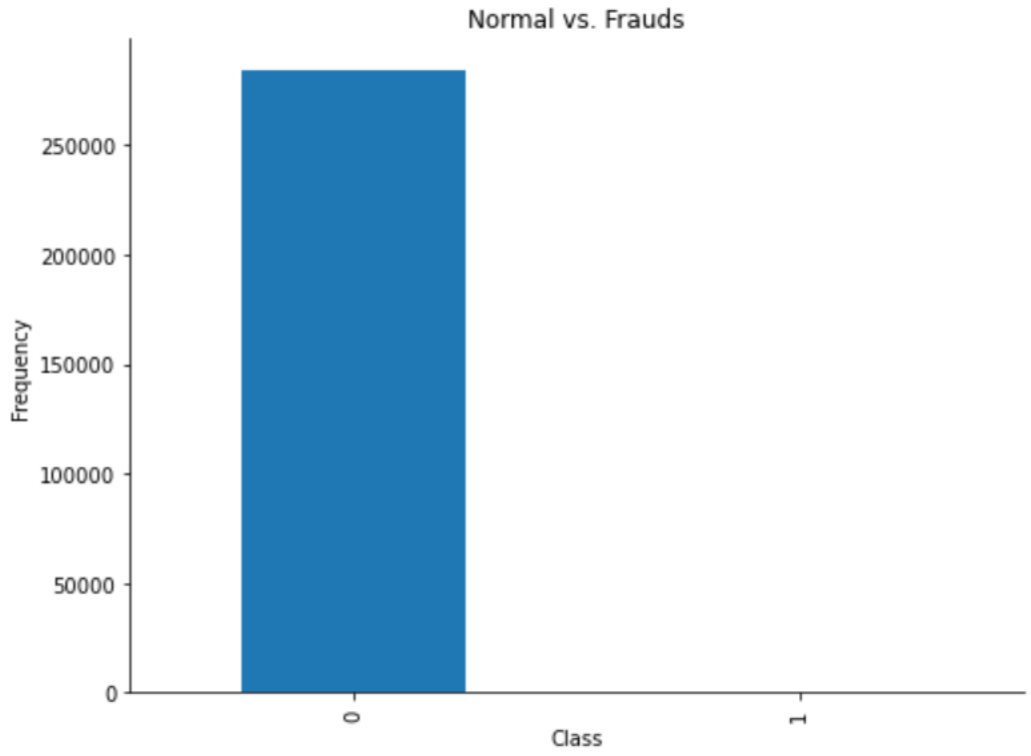

Figure 4. Correlations between the attributes in the unbalanced dataset

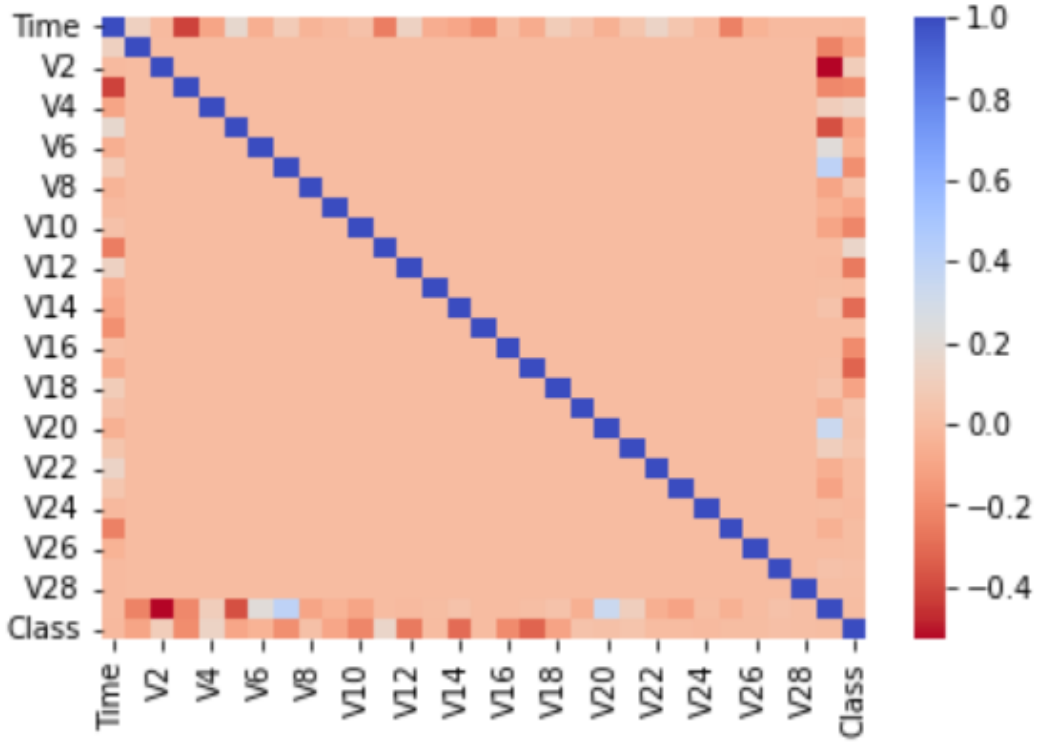

Table 1. Results from different algorithms

Resampling Method	Modeling Techniques	Parameters	Recall	Accuracy	F1-score
None	Logistic Regression	Default	0.58	1.00	0.69
	Logistic Regression	class_weight = 'balanced'	0.88	0.97	0.09
	XGBoost	Default	0.93	1.00	0.87
Random Undersampling	Logistic Regression	Default	0.87	0.95	0.06
	Logistic Regression	class_weight = 'balanced'	0.90	0.95	0.06
	Logistic Regression	penalty='l2'	0.90	0.96	0.08
	Decision Tree	Default	0.01	0.89	0.02
	Decision Tree	Max_depth =10	0.02	0.90	0.03
	Decision Tree	Max_depth =5	0.03	0.95	0.06
	Decision Tree	Max_depth =3	0.04	0.97	0.08
Random Oversampling	Logistic Regression	Default	0.90	0.97	0.08
	Logistic Regression	class_weight = 'balanced'	0.88	0.97	0.09
	Logistic Regression	penalty='l2'	0.90	0.97	0.08

	Decision Tree	Default	0.79	1.00	0.79
	Decision Tree	Max_depth =10	0.17	0.99	0.28
	Decision Tree	Max_depth =5	0.08	0.98	0.14
	Decision Tree	Max_depth =3	0.04	0.95	0.07
	Random Forest	Default	0.93	1	0.85
SMOTE	Logistic Regression	Default	0.88	0.98	0.13
	Logistic Regression	class_weight = 'balanced'	0.87	0.98	0.14
	Logistic Regression	penalty='l2'	0.88	0.98	0.16
	Decision Tree	Default	0.42	1.00	0.55
	Decision Tree	Max_depth =10	0.16	0.99	0.27
	Decision Tree	Max_depth =5	0.13	0.99	0.22
	Decision Tree	Max_depth =3	0.10	0.99	0.18
	Random Forest	**Default**	**0.88**	**1.00**	**0.85**
SMOTE and Undersampling	Logistic Regression	Default	0.61	1.00	0.65

Table 2. Hyperparameter tuning results

	C	Accuracy
0	0.001	0.401826
1	0.010	0.552511
2	0.100	0.607306
3	**1.000**	**0.614155**
4	10.000	0.602740
5	100.000	0.607306
6	1000.000	0.607306