Credit Card Fraud Detection

Sricharan Sridhar, Sanidhya Chaudhary, Adrija Ghosh

DEPARTMENT OF COMPUTER SCIENCE VIT CHENNAI
Kelambakkam - Vandalur Rd, Rajan Nagar, Chennai, Tamil Nadu 600127
sricharan.sridhar2020@vistudent.ac.in
sanidhya.chaduahry2020@vitstudent.ac.in
adrija.ghosh2020@viststudent.ac.in

***Abstract* - Credit card fraud is a significant problem for both financial institutions and customers. Fraudulent activities can lead to financial losses, negative impacts on credit scores, and identity theft. In recent years, with the increasing use of credit cards for online transactions, the frequency and complexity of fraud attempts have also increased. To combat this issue, various fraud detection techniques have been developed using machine learning algorithms. These techniques analyze patterns and anomalies in transaction data to identify potential fraud cases. This abstract explores the concept of credit card fraud detection and highlights the importance of developing effective fraud detection systems to protect customers and financial institutions from fraudulent activities.**

***Keywords* - credit card, fraud, sampling, hyper parameter tuning, machine learning model**

I. INTRODUCTION

As the world is rapidly moving towards digitization and money transactions are becoming cashless, the use of credit cards has rapidly increased. The fraud activities associated with it have also been increasing which leads to a huge loss to the financial institutions. Therefore, we need to analyze and detect the fraudulent transaction from the non-fraudulent ones. In this paper, we present a comprehensive review of various methods used to detect credit card fraud. These methodologies include Under sampling and Oversampling - Working with unbalanced data

The application of methods for data balancing, such as under sampling and oversampling techniques are widely used in these cases. Changing the sampling makes the algorithm more "sensitive" to fraudulent transactions.

Under sampling is the technique of removing major class records from the sample. In this case, it is necessary to remove random records from the legitimate class (No fraud), in order to obtain a number of records close to the amount of the minority class (fraud) in order to train the model.

Oversampling is exactly the opposite: it means adding minority class records (fraud) to our training sample, thus increasing the overall proportion of fraud records. There are methods to generate samples from the minority class, either by duplicating existing records or artificially generating others

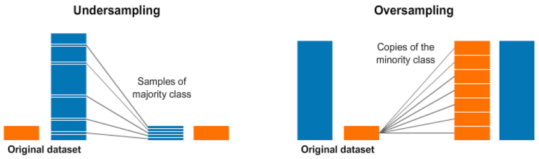

Fig. 1 Measurement of classifier performance through the accuracy evaluation metrics.

II. LITERATURE REVIEW

Credit card fraud detection is a critical area of research in the financial industry. With the increasing number of transactions and the complexity of fraudulent activities, developing effective fraud detection techniques has become more challenging. In this literature review, we will discuss some of the recent studies on credit card fraud detection.

One of the most popular techniques for credit card fraud detection is machine learning. Machine learning algorithms, such as decision trees, neural networks, and support vector machines, have been applied to credit card fraud detection. In a study by Bhattacharyya et al. (2020), they compared the performance of various machine learning algorithms on a credit card fraud detection dataset. They found that random forest, decision trees, and logistic regression had the highest accuracy in detecting fraudulent transactions.

Another technique that has gained attention in recent years is deep learning. Deep learning algorithms, such as convolutional neural networks and recurrent neural networks, have shown promising results in detecting credit

card fraud. In a study by Shen et al. (2019), they proposed a deep learning-based fraud detection system that outperformed traditional machine learning techniques.

Besides machine learning and deep learning, researchers have also explored other techniques for credit card fraud detection. For example, in a study by Panigrahi et al. (2021), they proposed a fraud detection system based on fuzzy logic. The system showed improved accuracy in detecting fraud cases compared to traditional machine learning techniques.

Overall, credit card fraud detection is a complex problem that requires continuous research and development. Machine learning, deep learning, and other techniques have shown promising results in detecting fraudulent activities. However, as fraudsters continue to develop new techniques, researchers need to keep developing and improving fraud detection systems to stay ahead of the game.

A. Problem Statement

Credit card fraud analysis plays a pivotal role in the credit card industry and is a critical component in safeguarding the security and integrity of financial transactions. By analyzing complex and often subtle patterns of data, fraud analysis can identify fraudulent activities and prevent unauthorized transactions, thereby mitigating the risk of financial loss and identity theft for customers. Furthermore, fraud analysis helps financial institutions minimize their exposure to financial losses, comply with regulatory requirements, and maintain the trust and confidence of their customers. The integration of fraud analysis can also help identify vulnerabilities in the credit card processing system, enabling institutions to develop and implement more effective security measures. Consequently, credit card fraud analysis represents a sophisticated and dynamic field, requiring advanced analytical tools, expert analysis, and constant vigilance in response to the continually evolving tactics and strategies of fraudsters.

B. Dataset

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. It contains only numeric input variables which are the result of a PCA transformation. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependent cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

III: METHODOLOGY

A. Handling Imbalanced Dataset

1) Random Undersampling

Random undersampling is a technique used to address the issue of class imbalance in a dataset. It involves reducing the size of the majority class by randomly removing samples, until the dataset becomes balanced.

The process of random undersampling is relatively straightforward. The first step is to identify the class that is in the majority and the class that is in the minority. Once this has been done, the majority class is randomly subsampled until its size is reduced to the same size as the minority class. This can be done using different approaches, such as randomly selecting a subset of the majority class, or removing all instances of the majority class that are not part of the minority class.

While random undersampling is a simple and easy-to-implement technique, there are some potential drawbacks to consider. One major concern is that by removing samples from the majority class, the resulting dataset may lose important information that is needed to accurately model the data. This can result in a decrease in the overall performance of the machine learning model. Additionally, random undersampling can lead to biased results if the removed samples are not truly representative of the majority class.

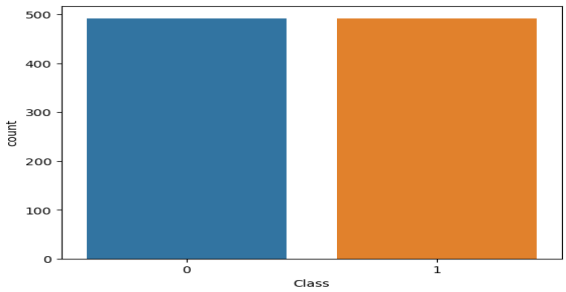

Fig. 2 Plotting classes proportion in random under sampled dataset : 492 of each class

2) Random Oversampling

Random oversampling is a technique used to address the issue of class imbalance in a dataset. It involves increasing the size of the minority class by randomly replicating samples until the dataset becomes balanced.

The process of random oversampling is relatively straightforward. The first step is to identify the class that is in the minority and the class that is in the majority. Once this has been done, the minority class is randomly oversampled until its size is increased to the same size as the majority class. This can be done using different approaches, such as randomly duplicating samples from the minority class, or creating new synthetic samples that are similar to existing samples in the minority class.

While random oversampling is a simple and easy-to-implement technique, there are some potential drawbacks to consider. One major concern is that by replicating samples from the minority class, the resulting dataset may become overfit to the minority class and result in a decrease in the overall performance of the machine learning model. Additionally, random oversampling can lead to biased results if the replicated samples are not truly representative of the minority class.

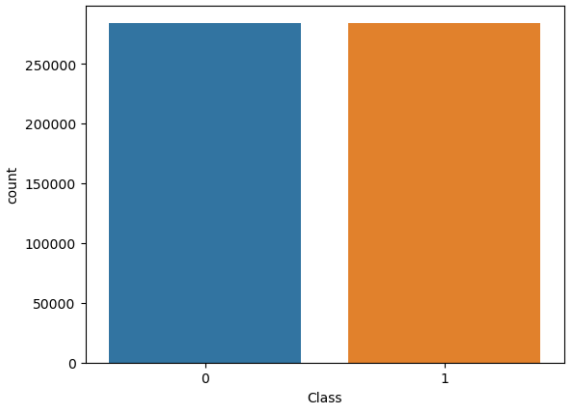

Fig. 3 Plotting classes proportion in random under sampled dataset : 284315 of each class

3) ADASYN oversampling

ADASYN is a more generic framework, for each of the minority observations it first finds the impurity of the neighborhood, by taking the ratio of majority observations in the neighborhood and k. Calculate the density distribution: ADASYN begins by calculating the density distribution of each minority class sample in the dataset. This is done by estimating the density of each minority sample based on the distances to its k nearest neighbors, and then normalizing the densities to sum up to 1.

Compute the imbalance ratio: ADASYN then computes the imbalance ratio of the dataset, which is the ratio between the number of majority class samples and the number of minority class samples.

Calculate the number of synthetic samples: For each minority class sample, ADASYN calculates the number of synthetic samples to be generated based on the local density distribution and the imbalance ratio. Specifically, the number of synthetic samples is calculated as:

#synthetic samples = (#majority samples - #minority samples) * density_i / sum(density)

where density i is the density of the i-th minority sample, and sum(density) is the sum of all densities of the minority samples.The number of synthetic samples is then rounded to the nearest integer.

Generate synthetic samples for each minority class sample, ADASYN generates the calculated number of synthetic samples by applying the SMOTE algorithm, but only to the minority samples in the k nearest neighbors that have lower densities than the current sample. This is done to ensure that the synthetic samples are generated in the regions of the feature space where the class boundary is more difficult to learn.

Specifically, for each minority sample, ADASYN selects its k nearest neighbors that belong to the minority class and have lower densities, and generates synthetic samples by applying the SMOTE algorithm. The number of synthetic samples is determined by the previously calculated number of synthetic samples.

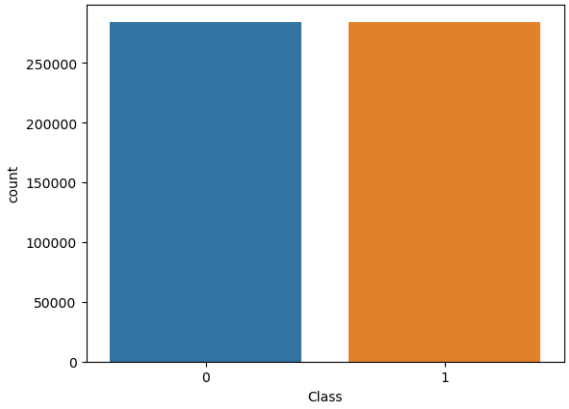

Fig 4. Plotting classes proportion in random under sampled dataset : 284338 of each class

4) SMOTE sampling

SMOTE (Synthetic Minority Over-sampling Technique) is a popular algorithm used for imbalanced data sets. It is a technique used to oversample the minority class in a data set by generating synthetic samples instead of replicating existing samples. This helps to balance the class distribution, which can improve the performance of machine learning models.

The basic idea behind SMOTE is to create new synthetic examples of the minority class by interpolating between existing minority class samples. This is done by selecting a random minority class example and then selecting one of its k nearest neighbors. A new synthetic example is then created by randomly interpolating between the two examples.

The interpolation process is done as follows: for each attribute in the selected minority class example, a new value is generated as a linear combination of the attribute values of the two examples. The weight of the linear combination is a random number between 0 and 1. This results in a new synthetic example that is different from both the selected minority class example and its nearest neighbor, but is still representative of the minority class.

The process of creating new synthetic examples continues until the desired ratio of minority class to majority class is achieved. This is often done by specifying a target percentage of the minority class in the final data set.

It is important to note that while SMOTE can improve the performance of machine learning models, it is not a silver bullet for imbalanced data sets. Careful consideration must be given to the choice of k (the number of nearest neighbors to consider), as well as other factors such as the choice of classifier and the evaluation metric used to measure performance. Additionally, SMOTE should only be used on the training data and not on the test data, as this can lead to overfitting.

5) Logistic regression

Logistic Regression is an algorithm used for classification purposes that estimates the likelihood of a target variable. The dependent variable in logistic regression is binary, meaning it has only two possible classes. Unlike linear regression, logistic regression models the output variable as a binary value using a sigmoid activation function. This function takes an input value, X, and returns an output value, S(X). To implement logistic regression, the algorithm requires both training and testing data. The regression coefficients are computed using the training data, and the sigmoid function is used to determine the relationship between the training and testing data. The Sigmoid function is given by:

$$S(X) = \frac{1}{1+e^{X}}$$

Here, X is the independent variable, and S(X) is the output. Algorithm for Logistic Regression:

- Input testing data and training data.
- Compute the regression coefficients of training data.
- Using sigmoid function find the relationship between training and testing data.
- Output the object's position.

Confusion matrix of Logistic regression classifier on random oversampled as it gives higher accuracy. Random Forest Classifier

The random forest is a popular algorithm due to its simplicity and diversity, and it is an ensemble technique that employs multiple decision trees. Each tree generates a set of predictions, and the final output prediction is determined by the majority vote of the classes, as depicted in figure 4. Rather than searching for the most important features to split, the random forest randomly selects a subset of features to determine the best feature for node splitting during the tree growing process.

This approach results in greater diversity, leading to a more robust model. Due to the low correlation between the models generated by the random forest, the ensemble predictions are more accurate than any individual predictions. This is due to the fact that although some trees may make errors, others will compensate for them.

6) Artificial Neural Network (ANN)

ANN is biologically inspired by human brain. The neurons are interconnected in the human brain like the same nodes are interconnected in artificial neural network, depicts the structure of ANN with input, output and hidden layers. Inputs are x1, x2…Xn and output is y. w1…wn are the weights associated with inputs x1…xn respectively. There are 5 hidden layers used in this neural network. The activation function used in our credit card fraud detection model is RELU.

The Proposed system uses the Artificial Neural Network to find the fraud in the credit card transactions. Performance is measured and accuracy is calculated based on prediction. And also classification algorithms such as Support vector machine and k-Nearest Neighbor are used to build a credit card fraud detection model. We compare all the three algorithms used in the experiment and made a decision that artificial neural networks predicts well than system developed using support vector machine and k-nearest neighbor algorithms. The dataset used in the experiment consist of 31 attributes out of which 30 attributes consist of information related to name, age, account information and so on and last attribute give the outcome of the transaction in either 0 or 1.

7) AdaBoost Boosting

The boosting method enhances the strength of weak learners, and the gradient boosting algorithm can be introduced by first discussing the AdaBoost algorithm. AdaBoost commences by training a decision tree with uniform weights assigned to each datapoint. After assessing the first tree's performance, it increases the weights of difficult-to-classify observations and decreases the weights of easy-to-classify ones. The second tree is then trained on this weighted data to improve upon the initial model's predictions, and the resulting model is the sum of Tree 1 and Tree 2. The classification error of this new 2-tree ensemble model is used to build a third tree that predicts the revised residuals. This iterative process continues for a

specific number of iterations, with the final ensemble model's predictions being the weighted sum of the previous models' predictions.

On the other hand, Gradient Boosting trains multiple models slowly, progressively, and sequentially. The primary distinction between AdaBoost and Gradient Boosting lies in how they detect the weak learners' shortcomings. AdaBoost identifies them using observations with high weights, while Gradient Boosting employs gradients in the loss function to achieve the same.

Adaptive Boosting or AdaBoost is utilized related to various calculations to improve execution. The outputs square measure combined by employing a weighted add that represents the combined output of the boosted classifier, which represents the combined output of the boosted classifier, i.e.

$$F\mathbf{t}(x) = f\mathbf{t}(x)$$

Where each ft is a classifier (weak learner) that profits the anticipated class concerning input x. Each feeble student gives a yield forecast, h(xi), for each preparation test. In each cycle t, the frail student is picked and is designated a coefficient, αt, so the preparation blunder whole, Et, of the subsequent t-stage helped classifier is limited,

$$E\mathbf{t} = \Sigma\ [F\mathbf{KO0}(x'') + \alpha\mathbf{K}h(x'')]$$

In which Ft–1(x) is a boosted classifier that is constructed in the preceding phase, E(F) represents the error function, and ft(x) = αth (x) is a weak learner who is considered for the final classifier. The vulnerable learners are squeezed concerning misclassified data samples by AdaBoost. However, it has susceptibility against noise and Outliers. AdaBoost has the potential for enhancing the individual outcomes from diverse algorithms till the classifier performs randomly.

B.Hyper Parameter Tuning

Hyperparameters that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.

1) Repeated k-Fold Cross-Validation for Model Evaluation

Repeated k-fold cross-validation provides a way to improve the estimated performance of a machine learning model. This involves simply repeating the cross-validation procedure multiple times and reporting the mean result across all folds from all runs. This mean result is expected to be a more accurate estimate of the true unknown underlying mean performance of the model on the dataset, as calculated using the standard error. We used 10 splits in our project.

2) Random Forest Classifier

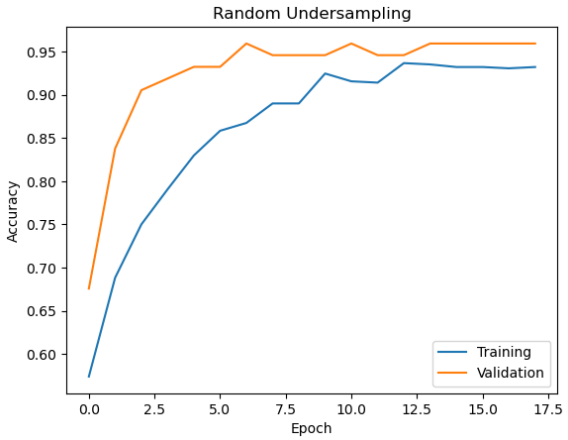

Fig 5. Random Undersampling model's accuracy plot

After using k fold cross validation via hyperparameter tuning, it able to avoid overfitting problem, as it clearly seen from above plot, both validation and training accuracies are increasing upto 95% best hyperparameters [activation=reLu, epochs=27, optimizer=adam] given by hyperparameter tuning by gridsearchcv

3) Logistic regression

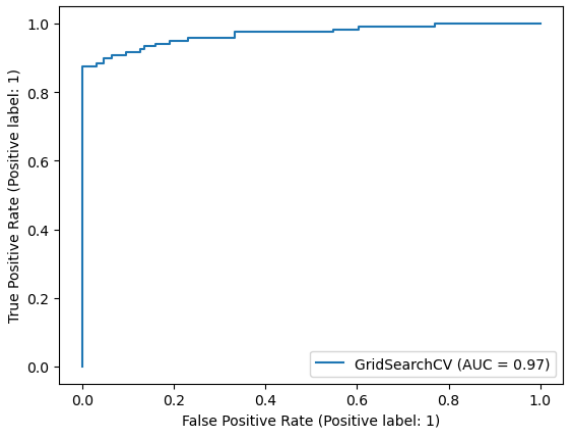

Fig 6. ROC Curve with score 0.97 of LR model on best hyperparameter given by hyperparameter tuning by gridsearchcv

3) AdaBoost

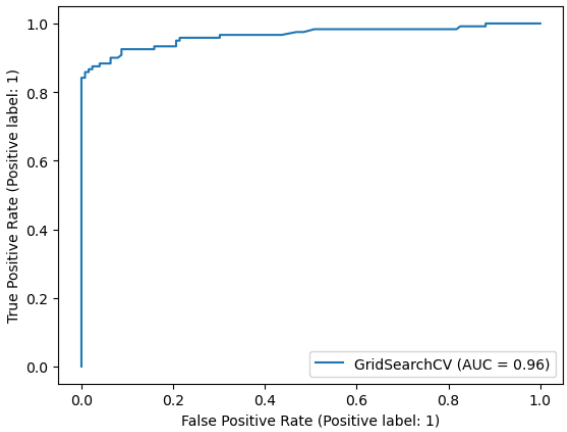

Fig 7. ROC Curve with score 0.96 of AdaBoost model on best hyperparameter given by hyperparameter tuning by gridsearchcv

IV: RESULTS & DISCUSSION

A. Correlation Matrix

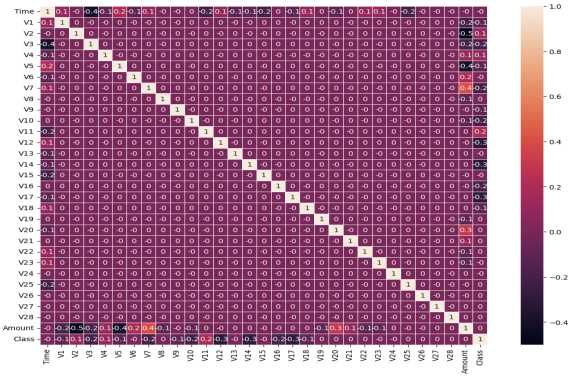

Fig 8. Heat map of all the features in the dataset

It can be inferred from the heat map that not many variables are correlated, only a few are correlated and even amongst them its a balanced split between negative and positive correlation. Furthermore, even those that are correlated are correlating with amount and class columns and no other parameters as one might expect out of sheer probability. We can see how V7 has a positive correlation and V2, and V4 have a negative correlation. Since there are no real class identifiers, we can guess the positive correlation variable might be income and the negative correlation variables maybe expenditure and something else related to money being spent.

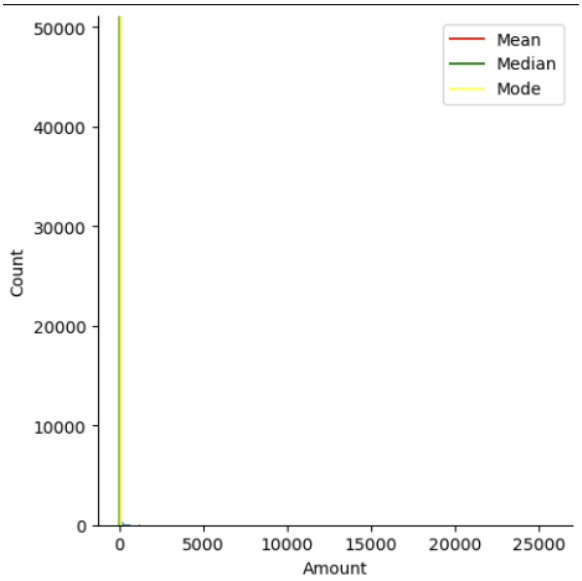

Fig 9. Amount vs Count diagram

Looking at the Amount vs Count column we can see how there is not a big spread in the values and it is a straight line with pretty much the mean, median and mode all in a close proximity. We assumed that this indicates that there is not a big spread, however, upon close inspection we got to know that the graph did not actually contain any values which led us to believe that there is something faulty with the code. Moving on, if we direct our attention to the Next graph which is, V1 vs Count we get a good idea about the spread of the parameter. Once again ths graph does not hold any meaning as the count variable and amount variable cannot be described by the measures of central tendency. Thus we did not proceed with the graph for the rest of the parameters that are there in the dataset.

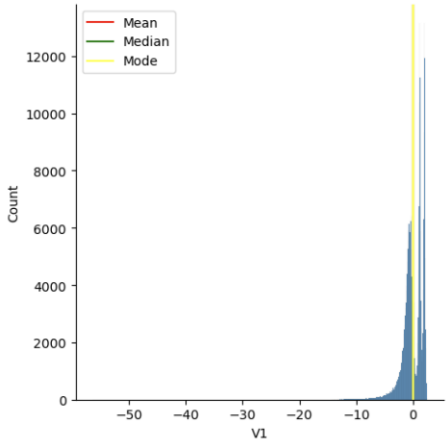

Fig 19. V1 vs Count diagram

The box and whisker plots are used to showcase the spread of a variable and it works as follows. Within the box there's supposed to be 50% of the data and outside the rest

of the 50%. Upon analysis of these plots it can be inferred how most of the elements do not necessarily have much of a spread and at the same time have some outliers deviating from the mean. If we take the plots for V1 till V15 we can see how the “whiskers” are very long yet the boxes are very small compared to a normal box and whiskers plot like that of Time. This actually shows how 50% of the data is actually near the mean and how there is not much of a spread within those values either indicated by the small box. The longer tails denote how there are many values that have deviated from the mean yet not so many of them as the boxes are very narrow. This is a sign that not many outliers are present in the data which not many people would infer after looking at the data. Are there many data points away from the mean? Yes. However, are they far enough from the mean to be called outliers? No. It could be said that if the data points are k*iqr away from the end of the box it could be considered as outliers which would be useful in determining the nature of a data point as outlier or not. An obvious example of an outlier would be the Amount, V22, and V26 plots where there are data points straying away from the mean and the 25% of the data collected on the whiskers.

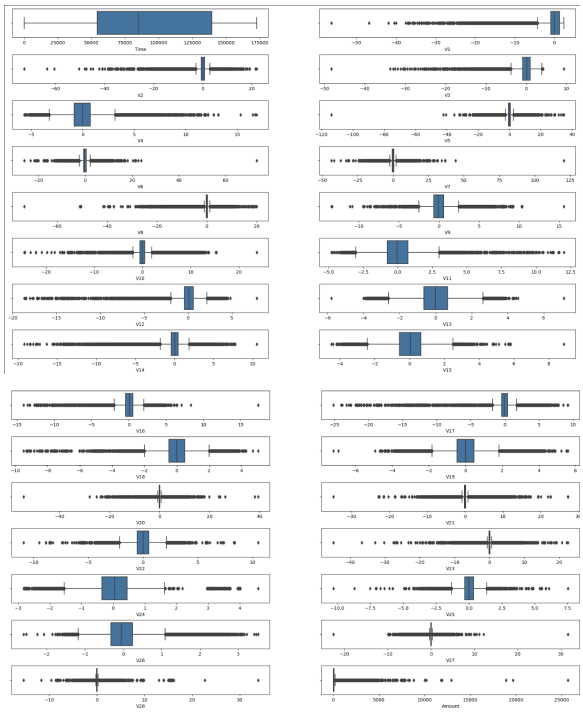

Fig 11. Box and whisker plot of all features

Moving onto a histogram plotting of the data we can infer how most of these are normal curves with not much of a right or left skew which indicates a normal distribution. However, there are many plots that are very narrow in width which do indicate how there is not much of a spread in these values. These are the parameters where even a slight deviation could mean so much considering the lack of spread over a wide range of values. An interesting instance to notice in these graphs is how in the Time graph there is a much wider spread than any other parameter, also how there are two dips in the graph and two peaks. This could mean something significant when predicting the class or amount variables, however the Heatmap suggests how the correlation is a mere 0.1 which immediately eradicates any such notion. Yet, it would be interesting and revealing to find out why exactly there was a dip when it came to certain numbers and more importantly why there was a peak when it came to a certain bunch of values namely from 25k-100k and 125k-175k. It is also not coincidental that both of these peaks are spread over a range of 75k. Considering that these are the seconds after an attack took place

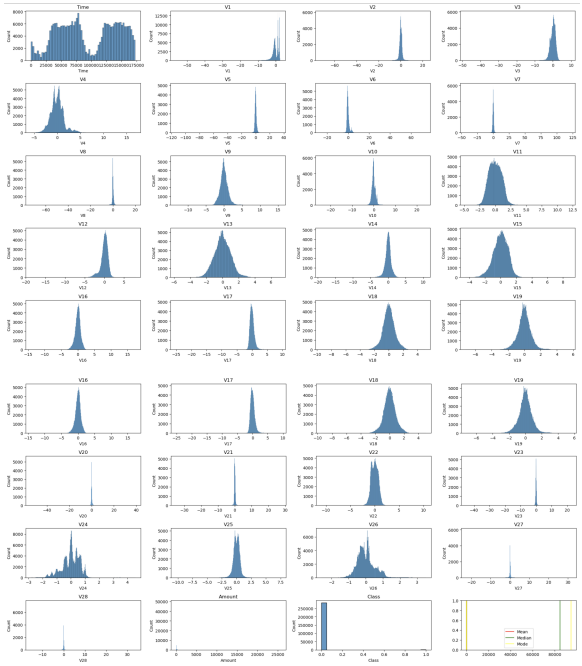

Fig 12. Feature vs Count Plot of all features

Now we look at the various class graphs that show the imbalances and balancing performed on them. We start with Random undersampled, Edited Undersample, then move onto the various Oversamplings such as random oversampling, Adasyn oversampling and SMOTE oversampling. When comparing each graph with the original Class vs Count Graph we can see how the amount of samples have changed. The undersamples usually take the majority class and then reduce it down to match the minority class. This can be inferred by how the count which was initially at around 25k for the majority class has dropped down to below 500 to match that of the minority class’s count. The exact opposite is what happens in the

oversampling techniques where the minority class is sampled to match the majority class. Both of these techniques have their disadvantages as follows:

B. Undersampling Disadvantages

- Loss of information of the majority class when reducing
- Reduction in Model Accuracy
- A slight bias towards the majority class

C. Oversampling Disadvantages

- Overfitting of the data
- Risk of data duplication
- Increased computation time

Whenever a decision has to be made on what sampling method to be used these disadvantages should be considered with the following model accuracies attained using the above sampling techniques.

Following tables are the accuracy value obtained from each of the models using the mentioned sampling techniques

0	original	0.999579
1	Random Undersampling	0.914634
2	Random Oversampling	0.999944
3	Adasyn Oversampling	0.999902
4	Smote Oversampling	0.999902

Fig 13. As it can be seen here the various results obtained for different datasets are mentioned for Random Forest Classifier.

Three sampling techniques out performed the model running without any sampling technique applied to it. They are Random Oversampling, Adasyn Oversampling and SMOTE oversampling. Random Oversampling also outperformed better than the other two oversampling methods. Moreover, It can also be inferred how Random under sampling significantly underperformed in comparison to the original model without any sampling. This could be due to the fact that most of the data is lost while reducing the sampling size which directly affects the accuracy of the model.

0	original	0.999354
1	Random Undersampling	0.947154
2	Random Oversampling	0.999550
3	Adasyn Oversampling	0.999691
4	Smote Oversampling	0.999669

Fig 145. As it can be seen here the various results obtained for the models are mentioned for Artificial Neural Network.

Once again the same 3 sampling techniques performed better than the original model with random undersampling performing better than it did for Random Forest Classifier. In this Adasyn Oversampling performed better than any other method that was trialed, however the results are nearly similar until the 10000th decimal place which goes to show how even though one may be better than the other the difference is very small. Even this small difference could play a big role when it come to a data set that is huge which should be noted.

0	original	0.999382
1	Random Undersampling	0.914634
2	Random Oversampling	0.951877
3	Adasyn Oversampling	0.894006
4	Smote Oversampling	0.950316

Fig 15. As it can be seen here the various results obtained for the models are mentioned for Logistic Regression.

This time around no sampling technique performed better than the original model itself. This can be attributed to the working mechanism of linear regression. When it comes to balanced and imbalanced datasets, linear regression may perform worse on a balanced dataset than on an imbalanced dataset because of the way the algorithm works. In a balanced dataset, there are roughly equal numbers of instances for each class or outcome being predicted, whereas in an imbalanced dataset, one class or outcome is much more prevalent than the others.

In a balanced dataset, linear regression may struggle to accurately model the relationship between the dependent variable and the independent variables because it will try to find a line that best fits all the data points, regardless of the class or outcome they belong to. This can result in a model that is not well-suited to predict the outcomes for either class.

On the other hand, in an imbalanced dataset, linear regression may perform better because it tends to be biased towards the majority class or outcome. This means that it will more accurately model the relationship between the dependent variable and the independent variables for the majority class, which is often the most important class in an imbalanced dataset.

0	original	0.999242
1	Random Undersampling	0.922764
2	Random Oversampling	0.963843
3	Adasyn Oversampling	0.939051
4	Smote Oversampling	0.966826

Fig 16. These are the obtained results for the Sampling Techniques mentioned for Adaboost model.

In the case of a balanced dataset, each class has roughly the same number of samples. As a result, each weak model trained by AdaBoost will have a similar accuracy rate for each class. However, when the dataset is imbalanced, AdaBoost may perform better because it can focus on learning the minority class, which is often the class of interest in many real-world applications.

When AdaBoost is applied to an imbalanced dataset, it places more emphasis on the minority class during the training process. This means that AdaBoost will try to reduce the number of false negatives (i.e., samples from the minority class that are misclassified as the majority class). As a result, AdaBoost can achieve better performance on the minority class, even if the overall accuracy of the model may not be as high as it would be on a balanced dataset.

On the other hand, when AdaBoost is applied to a balanced dataset, it may not be able to achieve the same level of performance as it can on an imbalanced dataset. This is because there is no clear minority class to focus on, and the algorithm will try to reduce the number of errors equally for both classes. This can result in a suboptimal model that is not specialized in any particular class.

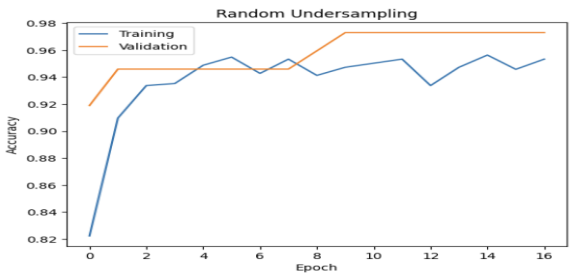

Fig 17. Random Sampling model accuracy

This graph depicts the accuracy of Random Undersampling after each epoch run. It can be observed how there is an increase as the epochs increase with just a few epochs that dip in terms of accuracy. However, if we look at the validation graph there is a step like increase which is different compared to the frenzy nature of the training data which can be explained by the difference in way it was trained during each epoch.

Overall the lines are indicating the increase in accuracy as the epochs increase which is a good sign.

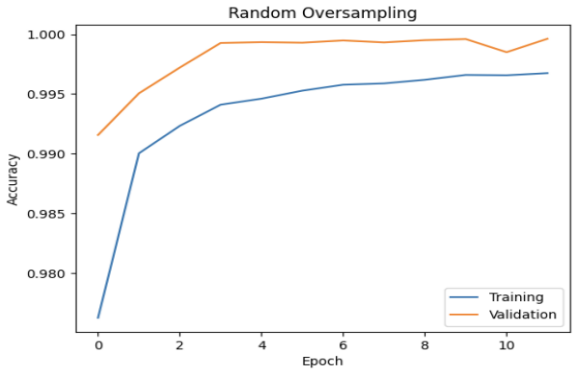

Fig 18. Random Sampling model accuracy

This graph depicts the accuracy of the Random Oversampling technique after each epoch run. This is the expected result for any graph as the validation results are better than the training results.

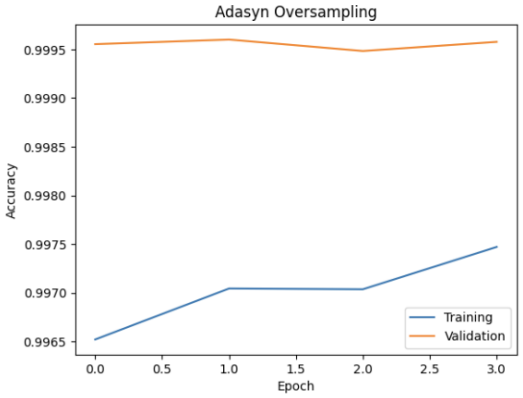

Fig 19. Adasyn Oversampling model accuracy

There might seem like there is a big difference between the two models, however, the difference between them is very small and in the 1000th of decimal places. This goes to show that ADASYN is quite effective without any preliminary training.

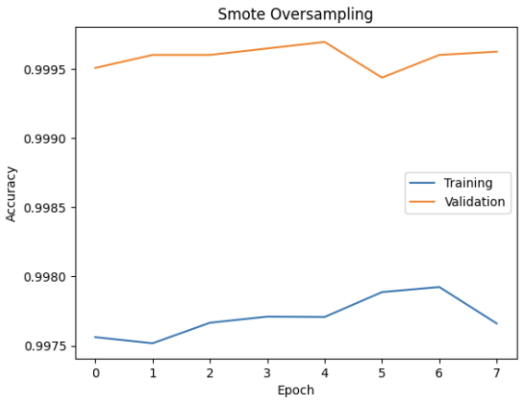

Fig 20. SMOTE oversampling model accuracy

There might seem like there is a big difference between the two models, however, the difference between them is very small and in the 1000th of decimal places. This goes to show that SMOTE is quite effective without any preliminary training. There is also a slight dip as the epochs increase.

V: CONCLUSION

In conclusion, credit card fraud is a serious problem that affects both consumers and financial institutions. Fortunately, there are various techniques like random forest technique, ANN and logistic regression methods that can be employed to detect and prevent credit card fraud. These include rule-based systems, anomaly detection, machine learning models, and more.

However, it is important to note that fraudsters are constantly evolving their tactics to circumvent detection systems, which makes it essential for financial institutions to remain vigilant and adapt their fraud detection strategies accordingly. Additionally, educating consumers about how to protect themselves from credit card fraud and encouraging them to monitor their accounts regularly can also help to prevent fraud.

Overall, credit card fraud detection is a dynamic field that requires a combination of technology, human expertise, and consumer awareness to effectively combat fraud and protect financial institutions and consumers alike.

ACKNOWLEDGMENT

It is our pleasure to express with deep sense of gratitude to Course faculty Dr. SA.Sajidha, Associate Professor Senior, SCOPE, Vellore Institute of Technology, Chennai, for her constant guidance, continual encouragement, understanding; more than all, she taught me patience in our endeavor.

Our association with her is not confined to academics only, but it is a great opportunity on our part to work with an intellectual and expert in the field of Information Retrieval and Organization.

Our sincere thanks to all the faculties and staff at Vellore Institute of Technology, Chennai, who helped us acquire the requisite knowledge. We would like to thank our parents for their support. It is indeed a pleasure to thank our friends who encouraged us to take up and complete this task.

REFERENCES

Ileberi, E., Sun, Y. & Wang, Z. A machine learning based credit card fraud detection using the GA algorithm for feature selection. *J Big Data* 9, 24 (2022). https://doi.org/10.1186/s40537-022-00573-8

Bin Sulaiman, R., Schetinin, V. & Sant, P. Review of Machine Learning Approach on Credit Card Fraud Detection. *Hum-Cent Intell Syst* 2, 55–68 (2022). https://doi.org/10.1007/s44230-022-00004-0

Husejinovic, A. (2020). Credit card fraud detection using naive Bayesian and C4.5 decision tree classifiers. Periodicals of Engineering and Natural Sciences, ISSN 2303-4521, 8 (1),

Baker, M.R., Mahmood, Z.N., Shaker, E.H. (2022). Ensemble learning with supervised machine learning models to predict credit card fraud transactions. Revue d'Intelligence Artificielle, Vol. 36, No. 4, pp. 509-518. https://doi.org/10.18280/ria.360401

Tiwari, P., Mehta, S., Sakhuja, N., Kumar, J., & Singh, A. K. (2021). Credit Card Fraud Detection using Machine Learning: A Study. *ArXiv*. /abs/2108.10005

Dornadula, V. N., & Geetha, S. (2019). Credit Card Fraud Detection using Machine Learning Algorithms. *Procedia Computer Science*, *165*, 631-641. https://doi.org/10.1016/j.procs.2020.01.057

Mansouri, T., Sadeghimoghadam, M., & Sahebi, I. G. (2021). A New Algorithm for Hidden Markov Models Learning Problem. *ArXiv*. /abs/2102.07112

Fraiman, J., Erviti, J., Jones, M., Greenland, S., Whelan, P., Kaplan, R. M., & Doshi, P. (2022). Serious adverse events of special interest following mRNA COVID-19 vaccination in randomized trials in adults. *Vaccine*, *40*(40), 5798-5805. https://doi.org/10.1016/j.vaccine.2022.08.036