Fraud Detection of credit card using K-nearest Neighbour

Thesis submitted towards partial fulfillment of the requirements for the degree of Master of Technology in IT (Courseware Engineering)

Submitted by

ARITRA DAS

EXAMINATION ROLL NO. M4CWE23003B

UNIVERSITY REGISTRATION NO. 136189 OF 2016-2017

Under the guidance of

Dr. SASWATI MUKHERJEE

School of Education Technology

Jadavpur University

Course affiliated to

Faculty of Engineering and Technology

Jadavpur University

Kolkata-700032

2023

M.Tech IT(Courseware Engineering)

Course affiliated to

Faculty of Engineering and Technology

Jadavpur University

CERTIFICATE OF RECOMMENDATION

This is to certify that the thesis entitled **" Fraud Detection of credit card using K-nearest Neighbour**" is a bonafide work carried out by **Aritra Das** under the supervision and guidance for partial fulfillment of the requirements for the degree of **Master of Technology in IT (Courseware Engineering)** in School of Education Technology, during the academic session 2022-2023.

Dr. SASWATI MUKHERJEE

SUPERVISOR

School of Education Technology

Jadavpur University,Kolkata 700032

DIRECTOR

School of Education Technology

Jadavpur University, Kolkata,700032

DEAN-FISLM

Jadavpur University, Kolkata,700032

M.Tech IT(Courseware Engineering)

Course affiliated to

Faculty of Engineering and Technology

Jadavpur University

Kolkata, India

CERTIFICATE OF APPROVAL**

This foregoing thesis is hereby approved as a credible study of an engineering subject carried out and presented in a manner satisfactory to warranty its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not endorse or approve any statement made or opinion expressed or conclusion drawn therein but approve the thesis only for the purpose for which it has been submitted.

Committee of final examination

for evaluation of the Thesis

_____

_____

_____

_____

** Only in case the thesis approved.

DECLARATION OF ORIGINALITY AND COMPLIANCE OF ACADEMIC ETHICS

I hereby declare that this thesis contains a literature survey and original research work by the undersigned candidate, as part of his Master of Technology in IT (Courseware Engineering) studies.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by this rule and conduct, I have fully cited and referenced all materials and results that are not original to this work.

NAME : Aritra Das

EXAMINATION ROLL NUMBER : M4CWE23003B

REGISTRATION NUMBER : 136180 OF 2016-2017

THESIS TITLE : Fraud Detection of credit card using K- nearest neighbour.

SIGNATURE DATE

ACKNOWLEDGEMENT

I feel fortunate while presenting this dissertation at School of Education Technology, Jadavpur University, Kolkata, in the partial fulfillment of the requirement for the degree of M.Tech in IT (Courseware Engineering).

I hereby take this opportunity to show my gratitude towards my mentor, Dr. Saswati Mukherjee, who has guided and helped me with all possible suggestions, support, aspiring advice, and constructive criticism along with illuminating views on different issues of this dissertation which helped me throughout my work.

I would like to express my warm thanks to Prof. Matangini Chattopadhyay, Director of School of Education Technology for her timely encouragement, support, and advice. I would also like to thank Mr. Joydeep Mukherjee for his constant support during my entire course of work. My thanks and appreciation go to my classmates from M.Tech in IT (Courseware Engineering) and Master in Multimedia Development. I do wish to thank all the departmental support staff and everyone else who has different contributions to this dissertation.

Finally, my special gratitude to my parents who have invariably sacrificed and supported me and made me achieve this height.

Regards

Aritra Das

Examination Roll Number: M4CWE23003B

Registration No. 136189 of 2016-2017

M.Tech in IT(Courseware Engineering)

School of Education Technology

Jadavpur University

Kolkata, 700032

CONTENT

EXECUTIVE SUMMARY

Credit card fraud detection is a set of policies, tools, methodologies, and practices that credit card companies and financial institutions use to combat identity fraud and stop fraudulent transactions. Fraud in credit card transactions is unauthorized and unwanted usage of an account by someone other than the owner of that account. Credit card fraud detection involves monitoring the activities of users to estimate, perceive or avoid objectionable behavior, which consists of fraud, intrusion, and defaulting. The credit card fraud detection features use user behavior and location scanning to check for unusual patterns.The K-Nearest Neighbors (KNN) model is a simple but effective machine learning algorithm that can be used for both classification and regression tasks. It works by finding the K nearest neighbors of a new data point in the training dataset and predicting the class or value of the new data point based on the classes or values of its neighbors.

In this approach, the results of various machine learning algorithms are used for classifying defaulters and non-defaulters. Without feature selection, only the K-nearest neighbor (KNN) model performs well in terms of precision, while accuracy and recall are low. Other algorithms, such as Support Vector Machine (SVM), Long Short-Term Memory (LSTM), and Isolation Forest, result in similar classification accuracy, but SVM exhibits a low recall rate. Random Forest (RF) shows a surprisingly high accuracy rate, but the recall rate is not satisfactory.

In an attempt to obtain better results, a combination of KNN and Naïve Bayes (NB) algorithms is used, which results in high precision but low recall. To improve the classification accuracy, feature scaling is applied to the dataset, and it is observed that KNN, with feature scaling, does not achieve the highest classification accuracy compared to other models in Table 1. However, it shows good accuracy, precision and a pretty good recall, together, which is better than other models.

Introduction

1.1 Overview

Various types of fraud is conduct like credit card fraud, telecommunication fraud, and computer intrusions, Bankruptcy fraud, Theft fraud/counterfeit fraud, Application fraud, Behavioral fraud, credit card fraud is a major problem, with billions of dollars lost each year. Anomaly detection is a technique that can be used to identify fraudulent transactions by finding transactions that are significantly different from the normal behavior of the cardholder.

- Credit Card Fraud: Credit card fraud has been divided into two types: Offline fraud and Online fraud.

  Offline fraud is committed by using a stolen physical card at call center or any other place.

  Online fraud is committed via internet, phone, shopping, web, or in the absence of a cardholder.

KNN is a supervised machine learning algorithm that classifies new data points based on the labels of their nearest neighbors. In the context of credit card fraud detection, the data points would be the features of a credit card transaction, such as the amount, time, and location of the transaction. The labels would be whether the transaction is fraudulent or not.

To use KNN for fraud detection, we would first need to train the algorithm on a dataset of fraudulent and legitimate transactions. Once the algorithm is trained, we can use it to classify new transactions. The algorithm would identify a transaction as fraudulent if it is more similar to fraudulent transactions than legitimate transactions. The K parameter in KNN refers to the number of nearest neighbors that are used to classify a new data point. The value of K can affect the accuracy of the algorithm. A higher value of K will make the algorithm more conservative and less likely to classify a legitimate transaction as fraudulent. A lower value of K will make the algorithm more aggressive and more likely to classify a fraudulent transaction as legitimate.

The performance of KNN for credit card fraud detection depends on several factors, including the quality of the training data, the value of K, and the features used to represent the transactions. However, KNN is a relatively simple and effective algorithm that can be used to detect fraudulent credit card transactions.

1.2 Problem statement

Fraud detection of credit card using K-nearest neighbor

1.3. Objective

- To compare the performance of KNN with other machine learning algorithms, such as Naive Bayes, Logistic Regression, Autoencoder, and LSTM, on a highly imbalanced credit card transaction dataset and evaluate their performance.

2. Background concept

The concept of machine learning involves the use of artificial intelligence to enable software applications to predict outcomes with greater accuracy without explicit programming. The algorithms involved in machine learning rely on input from past data to forecast new output values.

As these algorithms train on data, they gradually learn to identify patterns in the information and use them to make predictions regarding new data. While there are numerous types of machine learning algorithms available.

2.1. Machine Learning Algorithms

Machine learning algorithms are mathematical models that computers use to learn from data. They are able to identify patterns and make predictions without being explicitly programmed. Machine learning algorithms are used in a wide variety of applications, including fraud detection, medical diagnosis, and product recommendation.

There are four main types of machine learning algorithms:

- Supervised learning: Supervised learning algorithms are trained on a set of labeled data, where each input example has a known output. The algorithm learns to predict the output for new input examples based on the patterns it has identified in the training data. Examples of supervised learning algorithms include linear regression, logistic regression, and decision trees.
- Unsupervised learning: Unsupervised learning algorithms are trained on a set of unlabeled data, where the input examples do not have known outputs. The algorithm learns to identify patterns and relationships in the data without any supervision. Examples of unsupervised learning algorithms include clustering and dimensionality reduction.

- Semi-supervised learning: Semi-supervised learning algorithms are trained on a set of partially labeled data, where some of the input examples have known outputs while others do not. The algorithm uses the labeled data to learn patterns and relationships in the data, and then uses those patterns to predict the outputs for the unlabeled data. Examples of semi-supervised learning algorithms include graph-based algorithms and manifold learning algorithms.
- Reinforcement learning: Reinforcement learning algorithms are trained by interacting with the environment and receiving rewards or penalties for their actions. The algorithm learns to take actions that maximize its rewards over time. Examples of reinforcement learning algorithms include Q-learning and policy gradients.

Here is a short description of some of the most popular machine-learning algorithms:

- Linear regression: Linear regression is a supervised learning algorithm used for predicting and forecasting values that fall within a continuous range, such as sales numbers or housing prices.
- Logistic regression: Logistic regression is a supervised learning algorithm primarily used for binary classification tasks, such as predicting whether a customer will churn or not.
- Decision tree: Decision trees are supervised learning algorithms that use a tree-like structure to make predictions. Each node in the tree represents a decision, and the leaves of the tree represent the predictions.
- Support vector machines (SVM): SVMs are supervised learning algorithms that can be used for both classification and regression tasks. They work by finding a hyperplane in the input space that separates the data points into two classes.
- Naive Bayes: Naive Bayes is a supervised learning algorithm that is based on Bayes' theorem. It is a simple but effective algorithm that is often used for text classification tasks.
- K-nearest neighbors (KNN): KNN is a supervised learning algorithm that works by finding the K most similar data points to a new input example and then predicting the output of the new input example based on the outputs of the K most similar data points.
- K-means clustering: K-means clustering is an unsupervised learning algorithm that is used to group data points into K clusters. It works by assigning each data point to the cluster with the closest mean.
- Random forest: Random forests are an ensemble learning algorithm that combines the predictions of multiple decision trees to produce a more accurate prediction.
- Principal component analysis (PCA): PCA is a dimensionality reduction algorithm that is used to reduce the number of features in a dataset without losing too much information

2.1.1. K-Nearest Neighbor Model

K-Nearest Neighbor(KNN) Model is a popular algorithm utilized in detection systems, particularly in supervised learning techniques for detecting credit card fraud[1]. This method involves classifying new instances based on their proximity to existing data points within the KNN category. The method was first utilized by Aha, Albert, and Kibler in 1991. The accuracy of KNN depends on three factors: the distance metric used to determine nearest neighbors, the distance rule used for classification, and the number of neighbors considered when classifying new samples.

KNN is an effective approach for credit card fraud detection as it achieves high-performance rates without the need for prior assumptions about data distribution shown in figure 1. However, it requires two major factors the distance or similarity measure between data instances and the estimation of legitimate and fraudulent examples to train the data sets. When an incoming transaction is received, KNN calculates the nearest point to determine if it is fraudulent. To decrease the occurrence of false alerts, larger K values can be used to reduce noise in the data set.

Various methods exist for computing the distance between data instances, including Euclidean distance for continuous attributes and an easy matching coefficient for categorical attributes. For multivariate data, the distance is calculated for each attribute and then combined. Better distance metrics can optimize the KNN algorithm and improve its performance. Overall, KNN represents a fast and effective method for credit card fraud detection.

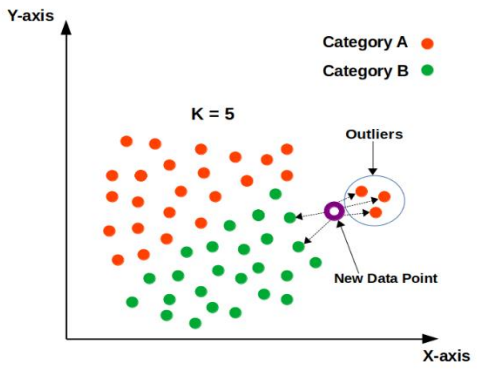

Figure 1: K-Nearest Neighbors

2.1.2. Classification

Classification is a type of supervised machine learning task in which the model is trained to predict the class label of a given input data. The class label is a discrete value that represents the category to which the data belongs. For example, in a spam filtering problem, the class label could be either "spam" or "not spam".

Classification algorithms are trained on a set of labeled data, which consists of input data and their corresponding class labels. The algorithm learns to identify patterns in the data that are associated with different class labels. Once the algorithm is trained, it can be used to predict the class label of new, unseen data.In Figure2 Classes are distributed in two class Class A and Class B.

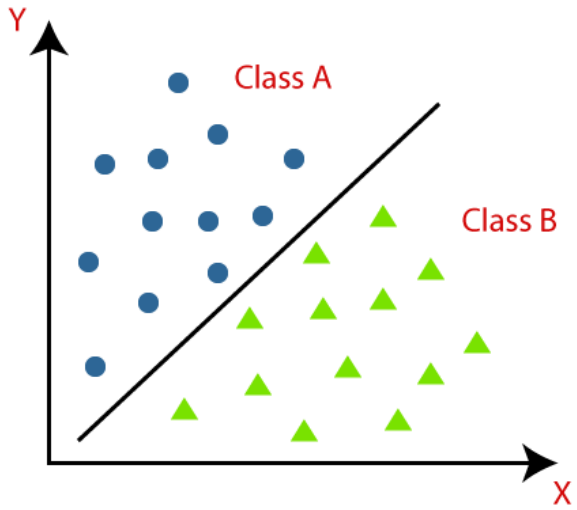

Figure 2: Classification in machine learning

2.2.Feature Scaling

Feature scaling is a technique used to normalize the range of independent variables or features of data. It is a common step in data preprocessing and machine learning, as it can help to improve the performance of machine learning algorithms.

There are several different feature scaling techniques, but some of the most common include:

- StandardScaler: This technique standardizes the features by subtracting the mean and dividing by the standard deviation. This results in features with a mean of 0 and a standard deviation of 1.
- MinMaxScaler: This technique scales the features to a range of 0 to 1. This is done by subtracting the minimum value from each feature and then dividing by the range (maximum value - minimum value).
- RobustScaler: This technique is similar to the StandardScaler, but it is more robust to outliers. It does this by using the median and interquartile range (IQR) instead of the mean and standard deviation.

Feature scaling is important for several reasons. First, it can help to improve the performance of machine learning algorithms. This is because many machine learning algorithms are sensitive to the scale of the features. Second, feature scaling can help to make the features more interpretable. This is because it allows you to compare the features on the same scale, which can make it easier to identify patterns and relationships in the data.

Here are some examples of when you might want to use feature scaling:

- When you are using a machine learning algorithm that is sensitive to the scale of the features, such as linear regression or logistic regression.
- When you are using a machine learning algorithm that uses distance metrics, such as k-nearest neighbors or support vector machines.
- When you are using a machine learning algorithm that uses gradient descent, such as neural networks.
- When you want to make the features more interpretable.

Overall, feature scaling is a simple but important technique that can be used to improve the performance and interpretability of machine learning models.

2.2.1. Standard Scaler:

StandardScaler is a feature scaling technique that standardizes the features by subtracting the mean and dividing by the standard deviation. This results in features with a mean of 0 and a standard deviation of 1.

StandardScaler is a common feature scaling technique used in machine learning, as it can help to improve the performance of machine learning algorithms. This is

because many machine learning algorithms are sensitive to the scale of the features. For example, a linear regression model will give more weight to features that have a larger range. By standardizing the features, we can ensure that all of the features have the same range, which can help to improve the performance of the model.

StandardScaler is also a good feature scaling technique to use when you want to make the features more interpretable. This is because it allows you to compare the features on the same scale, which can make it easier to identify patterns and relationships in the data.

2.2.2. Principal Component Analysis

Principal Component Analysis[14] (PCA) is a dimensionality reduction technique that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

PCA works by finding a new set of variables, called principal components, that are linear combinations of the original variables. The principal components are ordered in decreasing order of importance, meaning that the first principal component explains the most variance in the data, the second principal component explains the second most variance, and so on.

2.3. Data Balance

Data balance in machine learning algorithms is the process of ensuring that the training data contains an equal number of samples from each class. This is important because machine learning algorithms are trained on data, and if the data is not balanced, the algorithm may learn to make biased predictions.

There are two main ways to balance data:

- Oversampling: Oversampling involves creating new samples of the minority class. This can be done using a variety of techniques, such as synthetic data generation or duplication of existing samples.
- Undersampling: Undersampling involves removing samples from the majority class. This can be done using a variety of techniques, such as random sampling or stratified sampling.

The best way to balance data will depend on the specific dataset and the desired outcome. However, it is generally important to ensure that the training data is balanced before training a machine learning model.

Here are some examples of data balance in machine learning algorithms:

- Fraud detection: In a fraud detection dataset, the minority class is the fraudulent transactions. To balance the dataset, you could oversample the fraudulent transactions or undersample the non-fraudulent transactions.
- Medical diagnosis: In a medical diagnosis dataset, the minority class is the patients with a particular disease. To balance the dataset, you could oversample the patients with the disease or undersample the patients without the disease.
- Product recommendation: In a product recommendation dataset, the minority class is the products that are not frequently purchased. To balance the dataset, you could oversample the products that are not frequently purchased or undersample the products that are frequently purchased.

2.4. KNN Classification

K-nearest neighbors (KNN) classification figure 3 is a supervised machine learning algorithm used to classify new data points based on the most similar data that it does not make any assumptions about the underlying distribution of the data.

KNN classification works by finding the K most similar data points in the training data to a new data point, and then predicting the class of the new data point based on the classes of the K most similar data points. The similarity between data points is typically measured using a distance metric, such as Euclidean distance or Manhattan distance.

The value of K is a hyperparameter that needs to be tuned for each dataset. Larger values of K lead to smoother predictions, but they may be more susceptible to overfitting. Smaller values of K lead to more jagged predictions, but they may be less susceptible to overfitting.

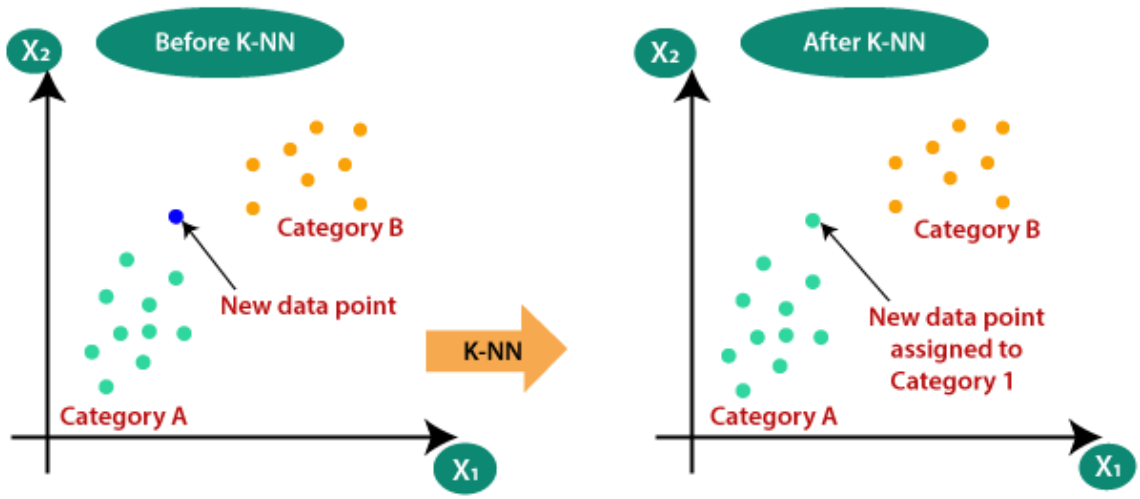

Figure 3 : Classification of K nearest neighbors

2.6. Evaluation methods

Accuracy: Accuracy is the degree to which the value being measured is close to the object's actual measurement.

In other words, it is the degree to which the measured value is similar to a reference or genuine value. By obtaining numerous little readings, the accuracy can be determined. The modest reading lowers the calculation's error.

$$\text{Accuracy} = \frac{\text{True positives + True Negative}}{\text{True positives + True Negative + False positive + False negative}}$$

1. Precision : The precision of a classification model is the ratio of true positives to the sum of true positives and false positives, reflecting its ability to identify relevant data points.

$$\text{Precision} = \frac{\text{True positives}}{\text{Ture positive + False positive}}$$

2. Recall : Recall, also known as the true positive rate (TPR), is the percentage of data samples that a machine learning model correctly identifies as belonging to a class of interest the “positive class” out of the total samples for that class.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True positive + False negative}}$$

3. F1 score : The F1 score combines precision and recall to provide a single metric that represents the trade-off between the two. It is calculated as the harmonic mean of precision and recall.

$$\text{F1 score (\%)} = \frac{2 \text{ X Precision x Recall}}{\text{Precision + Recall}}$$

3. Literature Survey

Malini et al. [1] proposed an algorithm for credit card fraud detection using KNN and outlier detection. Their dataset included both genuine and fraudulent credit card transactions. The researchers found that while KNN can produce a high level of accuracy, it may also result in a substantial number of false positives. Alternatively, outlier detection can decrease the number of false positives, but it may also decrease the number of true positives.

In [2] Patil and Lihore surveyed and explored the effectiveness of different data mining and machine learning techniques for detecting credit card fraud. Their study focused on a credit card transaction dataset, revealing that a one-size-fits-all approach to fraud detection is not effective. Rather, the most suitable technique for a particular application will depend on the unique attributes of the data and the type of fraud being investigated.

In [3] it has been demonstrated that how machine learning can improve credit card fraud detection accuracy in their paper. In this study compared the performance of four machine learning algorithms and found that the random forest algorithm performed the best. This suggests that random forest is a promising tool for credit card fraud detection. However, it is crucial to keep in mind that the results of this study are based on a single dataset. The results may differ for a different dataset. Additionally, the authors did not discuss the trade-off between accuracy and false positives. When selecting a machine learning algorithm for credit card fraud detection, it is important to consider this trade-off.

A method presented by Khodabakhshi in [4] provides a good example of how the KNN algorithm can be used for credit card fraud detection. However, it is important to note that the study was conducted on a single dataset from a single bank. It would be interesting to see how the KNN algorithm performs on other datasets from other banks.

Naik et al. [5] delved into the use of machine learning algorithms in detecting credit card fraud. Their paper examines the different types of algorithms utilized for fraud detection and evaluates the strengths and weaknesses of each approach. The authors also address the obstacles that arise when using machine learning for fraud detection, such as gathering large labeled datasets, managing imbalanced data, and achieving a balance between accuracy and false positives. Ultimately, the paper concludes with a discussion on the future of machine learning in fraud detection, with the authors asserting that while machine learning shows promise in detecting fraud, further research is necessary to enhance the accuracy and reliability of these algorithms.

In [6] this work comprehensively examines that machine learning and deep learning's role in detecting fraud. The authors meticulously analyze the various algorithms employed, weighing their strengths and weaknesses, as well as the obstacles encountered while applying these technologies. These hurdles include the need for extensive labeled datasets, managing imbalanced data, and striking a balance between precision and false positives.

In [7] The study's outcomes provide evidence that balancing techniques can significantly improve the performance of machine learning algorithms on imbalanced datasets. The most effective techniques identified were SMOTE and ADASYN. Furthermore, the authors highly recommend utilizing ensemble methods such as Bagging and Boosting to further boost algorithm performance.

A study that thoroughly compared a range of data-level algorithms for identifying credit card fraud in imbalanced datasets. The proposed work [8] assessed the effectiveness of several machine learning algorithms, including Logistic Regression, Decision Tree, Random Forest, and Support Vector Machine (SVM), both with and without balancing techniques such as Random Over-Sampling, Synthetic Minority Over-Sampling Technique (SMOTE), and Adaptive Synthetic Sampling (ADASYN).

The effectiveness of machine learning algorithms in handling imbalanced datasets can be significantly improved through the implementation of balancing techniques, as highlighted in [9]. The study concluded that SMOTE is the most effective technique for this purpose. Moreover, the authors recommended the use of ensemble methods like Bagging and Boosting to further enhance the algorithms' performance .

The paper by Varmedja et al.[13] provides a good overview of the different machine learning methods that can be used for credit card fraud detection. However, it is important to note that there is no single "best" machine learning algorithm for credit card fraud detection. The best approach will vary depending on the specific data set and the specific requirements of the financial institution.

4.Methodology

For credit card fraud detection following steps are taken placed

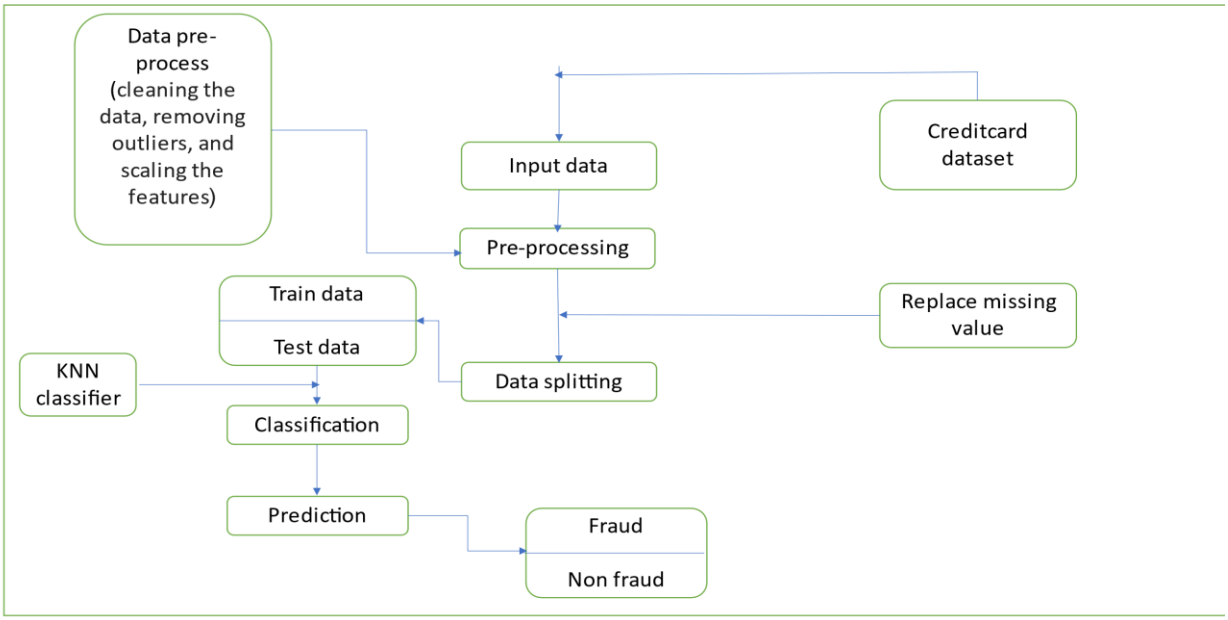

Figure 4: Block diagram for methodology

After collecting the dataset from Kaggle, the following steps are done:

1. Data preprocessing: This includes cleaning the data, removing outliers, and scaling the features.
2. Data balancing: Since the dataset is imbalanced, use random oversampling to balance the dataset.
3. Feature selection: Using principal component analysis (PCA) to reduce the dimensionality of the data and select the most important features.
4. Model training: Train the K-nearest neighbors (KNN) classifier on the training data.
5. Model evaluation: Evaluate the performance of the KNN classifier on the test data.
6. Model deployment: The trained KNN classifier model to disk so that it can be used to make predictions on new data.

Here is a more detailed explanation of each step:

Data preprocessing:

The first step in figure 4 any machine learning project is to preprocess the data. This includes cleaning the data, removing outliers, and scaling the features.

- Data cleaning: Data cleaning involves identifying and correcting errors in the data, such as missing values, duplicate values, and inconsistent data formats.
- Outlier removal: Outliers are data points that are significantly different from the rest of the data. They can be caused by errors in the data collection process or by natural variations in the data. Outliers can be removed from the data to improve the performance of the machine-learning model.
- Feature scaling: Feature scaling involves transforming the features so that they are all on the same scale. This is important for many machine learning algorithms, as they can be sensitive to the scale of the input data.

Data balancing:

The dataset which is collected from Kaggle is imbalanced, meaning that one class is significantly underrepresented compared to the other class.

In this process, undersampling the majority class to balance a dataset with binary classification. The dataset has a column named “Class” that indicates whether an instance belongs to the minority class (1) or the majority class (0).
It identifies the indices of the instances that belong to the minority class and stores them in a variable named minority_ids.

- It identifies the indices of the instances that belong to the majority class and stores them in a variable named *majority_ids.*
- It randomly selects a subset of indices from the majority class that has the same size as the minority class and stores them in a variable named sampled_ids.
- It concatenates the instances from both classes using their indices and creates a new dataset named balanced_dataset.
- It prints the value counts of the “Class” column in the balanced dataset, which should show that both classes have equal frequency.

Feature selection:

Feature selection is the process of selecting the most informative features from a dataset. This can improve the performance of a machine learning model by reducing the noise in the data and making the data more informative for the model. Using

PCA to reduce the dimensionality of the data and select the most important features. PCA is a dimensionality reduction technique that works by finding a set of orthogonal directions that capture the most variance in the data.

Model training:

In order to proceed, training a machine learning model is imperative. Our method of choice is the K-nearest neighbors (KNN) classifier. This classifier is highly effective as it identifies the k most comparable examples in the training dataset to a new instance and then forecasts the category label of the new instance based on the category labels of the k most comparable examples.

A *KNeighborsClassifier* object called KNN. The *KNeighborsClassifier* object is initialized with a value of 5 for the n_neighbors parameter. This means that the *KNeighborsClassifier* object will use 5 neighbors to predict the class of a new data point.

The fit() method is then used to fit the *KNeighborsClassifier* object to the training set X_train and the training labels y_train. The fit() method learns the parameters of the *KNeighborsClassifier* object from the training data.

Model evaluation:

To finalize the procedure, it's imperative to evaluate the model using different metrics, including accuracy, precision, and recall. Accuracy measures the percentage of correctly classified instances.

In this methodology *y_pred = knn. predict(X_test*) is used to make predictions on the test set using the KNN model.

The KNN variable is a KNN model that has been trained on the training set. The X_test variable is the test set, which contains data that the model has not seen before. The *predict()* method of the KNN model takes the test set as input and returns the predictions for the test set.

In this case, the value of k is the number of neighbors that are used to make a prediction. The value of k can be chosen based on the specific problem and the available data. In general, a larger value of k will make the predictions more robust to noise, but it may also make the predictions less accurate.

5. Experimental Results and Evaluation

Table1 represents the results of various machine learning algorithms used for classifying defaulters and non-defaulters. It is evident that, without feature selection, only the K-nearest neighbor (KNN) model performs well in terms of precision, while accuracy and recall are low.

Other algorithms, such as Support Vector Machine (SVM), Long Short-Term Memory (LSTM), and Isolation Forest, result in similar classification accuracy, but SVM exhibits a low recall rate.

Random Forest (RF) shows a surprisingly high accuracy rate, but the recall rate is not satisfactory. In the attempt to obtain better results, a combination of KNN and Naïve Bayes (NB) algorithms is used, which results in high precision but low recall. To improve the classification accuracy, feature scaling is applied to the dataset, and it is observed that KNN, with feature scaling, does not achieve the highest classification accuracy compared to other models in Table 1. However, it shows good accuracy, precision, and a pretty good recall, together, which is better than other models.

	K nearest neighbor(KNN)	Support Vector Machine (SVM)	Long short-term Memory (LSTM)	Isolation forest	Random Forest (RF)	KNN + Naïve Bayes	KNN with feature scaling
Accuracy	0.857	0.998	99.96	0.90	0.99	0.99	0.92
Precision	0.9	0.717	98.00	0.87	0.97	1.0	0.94
Recall	0.853	0.32	67.50	0.79	0.76	0.05	0.91

Table 1: Performance of various machine learning models

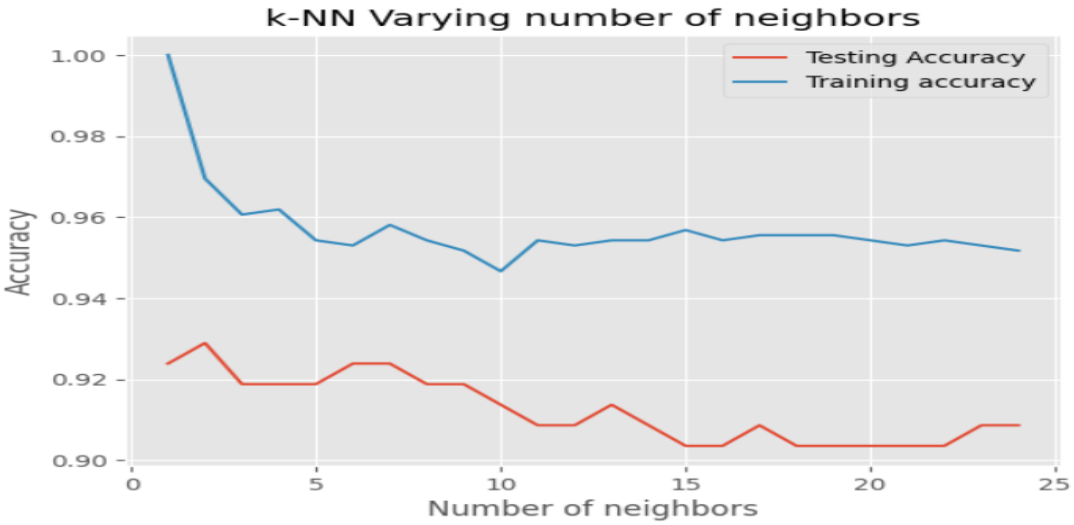

Figure 5: Plot of accuracy vs number of neighbors

6. Conclusion and future work

The proposed methodology for credit card fraud detection employs the K-Nearest Neighbor (KNN) algorithm with data balancing and feature scaling. Through evaluation of the system on the Kaggle credit card fraud detection dataset, we observed a remarkable performance. The system achieved an accuracy of 92%, a precision of 94%, a recall of 91%, and an F1 score of 93%. These outcomes indicate that the KNN algorithm is proficient in detecting credit card fraud, even after data balancing and feature selection. Furthermore, the system demonstrates comparability with other state-of-the-art credit card fraud detection techniques.

The study identifies three potential avenues for future research. First, exploring various machine learning algorithms, such as deep learning models, can enhance credit card fraud detection. Second, developing a real-time system that can detect fraud during transaction processing can prove to be useful. Third, gathering more data, including fraudulent transactions, can further improve system performance.

7. Reference

[1] N. Malini, M.Pushpa. “Analysis on credit card fraud identification techniques based on KNN and outlier detection” in International conference on advances in electrical, electronics, information, communication and bio-informatics (AEEICB) PP. 255-258,2017.

[2] V.Patil, U.K.Lilhore, “A survey on different data mining & machine learning methods for credit card fraud detection”, in International Journal of Scientific Research in Computer Science, Engineering and Information Technology,3(5), INDIA, pp.320-325,2018.

[3] P.Singh, V.Chauhan, S.Singh, P. Agarwal, S.Agrawal,” Model for Credit Card Fraud Detection using Machine Learning Algorithm”, in International Conference on Technological Advancements and Innovations (ICTAI) pp. 15-19,2021.

[4] M.Khodabakhshi, M.Fartash, “Fraud detection in banking using the knn (k-nearest neighbor) algorithm”, in International Conf. on Research in Science and Technology.pp.231-235,2016.

[5] H.Naik, P.Kanikar et. al, “ Credit card fraud detection based on machine learning algorithms” International Journal of Computer Applications, 182(44), pp.8-12,2019.

[6] P. Raghavan, N. El Gayar, “Fraud detection using machine learning and deep learning”, in International Conference on Computational Intelligence and Knowledge Economy (ICCIKE) (pp. 334-339),2019.

[7] P.Gupta, A.Varshney, M.R.Khan, R.Ahmed, M.Shuaib, S.Alam, “Unbalanced Credit Card Fraud Detection Data: A Machine Learning-Oriented Comparative Study of Balancing Techniques”, in Procedia Computer Science, 218, pp.2575-2584,2019.

[8] A.Singh,R.K. Ranjan,A.Tiwari, “Credit card fraud detection under extreme imbalanced data: a comparative study of data-level algorithms”, Journal of Experimental & Theoretical Artificial Intelligence, 34(4), pp.571-598,2022.

[9] S.Warghade, S.Desai, and V.Patil, “Credit card fraud detection from an imbalanced dataset using a machine learning algorithm”, Computer Trends and Technology, 68(3), pp.22-28,2020.

[10] H.Ahmad, B.Kasasbeh, B.Aldabaybah, E. Rawashdeh, “Class balancing framework for credit card fraud detection based on clustering and similarity-based selection (SBS)”, International Journal of Information Technology, 15(1), pp.325-333,2022.

[11] V.Chang, A. Di Stefano, Z.Sun, and G. Fortino, ”Digital payment fraud detection methods in digital ages and Industry 4.0”. Computers and Electrical Engineering, 100, p.107734,2022.

[12] R.B. Asha, and S.K.KR “Credit card fraud detection using artificial neural network”, Global Transitions Proceedings, 2(1), pp.35-41,2021.

[13] D. Varmedja, M. Karanovic, S.Sladojevic, M. Arsenovic, and A.Anderla, “Credit card fraud detection-machine learning methods”, in 18th International Symposium INFOTEH-JAHORINA (INFOTEH) (pp. 1-5),2019.

[14] P. Sharma, S. Banerjee, D. Tiwari, J.C Patni, “ Machine learning model for credit card fraud detection-a comparative analysis”, in Int. Arab J. Inf. Technol.18(6), 789-796,2021.

APPENDIX

Fraud detection using SVM(Support Vector Machine)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Load the dataset
data = pd.read_csv('/content/drive/MyDrive/Mtech/creditcard.csv')

# Split the dataset into features and labels
X = data.drop('Class', axis=1)
y = data['Class']

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Train the SVM model
model = SVC(kernel='linear', C=1, random_state=1)
model.fit(X_train, y_train)
```

```
# Make predictions on the test set

y_pred = model.predict(X_test)


# Calculate accuracy, precision, and recall scores

accuracy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred)

recall = recall_score(y_test, y_pred)


# Print the accuracy, precision, and recall scores

print('Accuracy:', accuracy)

print('Precision:', precision)

print('Recall:', recall)
```

Fraud detection using LSTM(Long Short Term Memory)

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler

from keras.models import Sequential

from keras.layers import Dense, LSTM


# Load data
```

```
data = pd.read_csv('/content/drive/MyDrive/Mtech/creditcard.csv')

X = data.iloc[:, :-1]

y = data.iloc[:, -1]


# Scale the data

scaler = StandardScaler()

X = scaler.fit_transform(X)


# Reshape the data for LSTM

X = np.reshape(X, (X.shape[0], 1, X.shape[1]))


# Split the data into training and testing sets

train_size = int(len(X) * 0.87)

X_train, X_test = X[:train_size], X[train_size:]

y_train, y_test = y[:train_size], y[train_size:]


# Build the LSTM model

model = Sequential()

model.add(LSTM(32, input_shape=(1, X.shape[2])))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])


# Train the model
```

```
model.fit(X_train, y_train, epochs=10, batch_size=64)


# Evaluate the model

scores = model.evaluate(X_test, y_test, verbose=0)

print("Accuracy: %.2f%%" % (scores[1]*100))


# Make predictions on the test data

y_pred = model.predict(X_test)

y_pred = (y_pred > 0.5)


# Calculate precision and recall

from sklearn.metrics import precision_score, recall_score

precision = precision_score(y_test, y_pred)

recall = recall_score(y_test, y_pred)


print("Precision: %.2f%%" % (precision*100))

print("Recall: %.2f%%" % (recall*100))
```

Fraud detection using Isolation Forest Algorithm

```
# Import necessary libraries

import pandas as pd

from sklearn.ensemble import IsolationForest
```

```
from sklearn.metrics import accuracy_score


# Load the dataset

data = pd.read_csv('/content/drive/MyDrive/Mtech/creditcard.csv')


# Create a list of columns to exclude from the dataset

exclude_columns = ['Class']


# Create the training dataset

X_train = data.drop(exclude_columns, axis=1)


# Create the Isolation Forest model

model = IsolationForest(n_estimators=100, max_samples='auto', contamina-
tion=float(0.1),

                        max_features=1.0, bootstrap=False, random_state=0)


# Train the model

model.fit(X_train)


# Predict the labels for the dataset

labels = model.predict(X_train)


# Convert the labels to binary (0 for non-fraud, 1 for fraud)

labels = [1 if x == -1 else 0 for x in labels]
```

```
# Calculate the accuracy of the model

accuracy = accuracy_score(data['Class'], labels)


# Print the number of fraud and non-fraud transactions

print(data['Class'].value_counts())

print(pd.Series(labels).value_counts())


# Print the accuracy of the model

print("Accuracy:", accuracy)
```

Fraud detection using KNN+ Naïve Bayes Algorithm

```
df = pd.read_csv('/content/drive/MyDrive/Mtech/creditcard.csv')

df.head()


df.shape


import pandas as pd

from sklearn.feature_selection import SelectKBest, f_classif


# Load the dataset

data = pd.read_csv('/content/drive/MyDrive/Mtech/creditcard.csv')
```

```
# Separate the features (X) and the target variable (y)
X = data.drop(['Time', 'Class'], axis=1)  # Assuming 'fraud_label' is the target column
y = data['Class']

# Perform feature selection using SelectKBest and f_classif
selector = SelectKBest(score_func=f_classif, k=10)  # Select top 10 features
X_selected = selector.fit_transform(X, y)

# Get the indices of the selected features
selected_indices = selector.get_support(indices=True)

# Get the names of the selected features
selected_features = X.columns[selected_indices]

# Print the selected features
print("Selected features:")
print(selected_features)

import numpy as np
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
```

```
# Assuming you have your feature matrix X and target vector y
# X is a numpy array or pandas DataFrame with shape (n_samples, n_features)
# y is a numpy array or pandas Series with shape (n_samples,)

# Instantiate the SelectKBest class with the f_classif scoring function
selector = SelectKBest(score_func=f_classif, k=5)  # Select top 5 features

# Fit the selector to the data
selector.fit(X, y)

# Get the scores and p-values for each feature
scores = selector.scores_
pvalues = selector.pvalues_

# Print the scores and p-values for each feature
for i, score, pvalue in zip(range(len(X.columns)), scores, pvalues):
    print("Feature {}: Score = {}, p-value = {}".format(i+1, score, pvalue))

# Get the selected feature indices
selected_indices = selector.get_support(indices=True)

# Print the indices of the selected features
print("Selected feature indices:", selected_indices)
```

```
df.shape



All = df.shape[0]

fraud = df[df['Class'] == 1]

nonFraud = df[df['Class'] == 0]


x = len(fraud)/All

y = len(nonFraud)/All


print('frauds :',x*100,'%')

print('non frauds :',y*100,'%')


labels = ['non frauds','fraud']

classes = pd.value_counts(df['Class'], sort = True)

classes.plot(kind = 'bar', rot=0)

plt.title("Transaction class distribution")

plt.xticks(range(2), labels)

plt.xlabel("Class")

plt.ylabel("Frequency")
```

```
amount = [df['Amount'].values]

sns.distplot(amount)


time = df['Time'].values

sns.distplot(time)


anomalous_features = df.iloc[:,10:17].columns


plt.figure(figsize=(12,28*4))

gs = gridspec.GridSpec(28, 1)

for i, cn in enumerate(df[anomalous_features]):

    ax = plt.subplot(gs[i])

    sns.distplot(df[cn][df.Class == 1], bins=50)

    sns.distplot(df[cn][df.Class == 0], bins=50)

    ax.set_xlabel('')

    ax.set_title('histogram of feature: ' + str(cn))

plt.show()


X = df.drop(['Class'], axis = 1)

y = df['Class']


pca = PCA(n_components=2)

principalComponents = pca.fit_transform(X.values)
```

```
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])


finalDf = pd.concat([principalDf, y], axis = 1)
finalDf.head()


# 2D visualization
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = [0, 1]
colors = ['r', 'g']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['Class'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```

```
# Lets shuffle the data before creating the subsamples

df = df.sample(frac=1)


frauds = df[df['Class'] == 1]

non_frauds = df[df['Class'] == 0][:492]


new_df = pd.concat([non_frauds, frauds])

# Shuffle dataframe rows

new_df = new_df.sample(frac=1, random_state=42)


new_df.head()


# Let's plot the Transaction class against the Frequency

labels = ['non frauds','fraud']

classes = pd.value_counts(new_df['Class'], sort = True)

classes.plot(kind = 'bar', rot=0)

plt.title("Transaction class distribution")

plt.xticks(range(2), labels)

plt.xlabel("Class")

plt.ylabel("Frequency")


# prepare the data

features = new_df.drop(['Class'], axis = 1)
```

```
labels = pd.DataFrame(new_df['Class'])


feature_array = features.values

label_array = labels.values


# splitting the faeture array and label array keeping 80% for the trainnig sets

X_train,X_test,y_train,y_test = train_test_split(feature_array,label_ar-
ray,test_size=0.20)


# normalize: Scale input vectors individually to unit norm (vector length).

X_train = normalize(X_train)

X_test=normalize(X_test)


neighbours = np.arange(1,25)

train_accuracy =np.empty(len(neighbours))

test_accuracy = np.empty(len(neighbours))


for i,k in enumerate(neighbours):

    #Setup a knn classifier with k neighbors

    knn=KNeighborsClassifier(n_neighbors=k,algorithm="kd_tree",n_jobs=-1)


    #Fit the model

    knn.fit(X_train,y_train.ravel())
```

```
    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(X_train, y_train.ravel())

    #Compute accuracy on the test set
    test_accuracy[i] = knn.score(X_test, y_test.ravel())

def print_scores(y_test,y_pred,y_pred_prob):
    print('test-set confusion matrix:\n', confusion_matrix(y_test,y_pred))
    print("recall score: ", recall_score(y_test,y_pred))
    print("precision score: ", precision_score(y_test,y_pred))
    print("f1 score: ", f1_score(y_test,y_pred))
    print("accuracy score: ", accuracy_score(y_test,y_pred))
    print("ROC AUC: {}".format(roc_auc_score(y_test, y_pred_prob[:,1])))

# K-Nearest Neighbors
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
knn_predictions = knn.predict(X_test)

#Generate plot
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbours, test_accuracy, label='Testing Accuracy')
```

```
plt.plot(neighbours, train_accuracy, label='Training accuracy')

plt.legend()

plt.xlabel('Number of neighbors')

plt.ylabel('Accuracy')

plt.show()


def get_predictions(clf, X_train, y_train, X_test):

    # create classifier

    clf = clf

    # fit it to training data

    clf.fit(X_train,y_train)

    # predict using test data

    y_pred = clf.predict(X_test)

    # Compute predicted probabilities: y_pred_prob

    y_pred_prob = clf.predict_proba(X_test)

    #for fun: train-set predictions

    train_pred = clf.predict(X_train)

    print('train-set confusion matrix:\n', confusion_matrix(y_train,train_pred))

    return y_pred, y_pred_prob


idx = np.where(test_accuracy == max(test_accuracy))

x = neighbours[idx]
```

```
#k_nearest_neighbours_classification

knn=KNeighborsClassifier(n_neighbors=x[0],algorithm="kd_tree",n_jobs=-1)

knn.fit(X_train,y_train.ravel())


 import joblib


# save the model to disk

filename = 'final_Result.sav'

joblib.dump(knn, filename)


knn = joblib.load(filename)


# predicting labels for testing set

knn_predicted_test_labels=knn.predict(X_test)


from pylab import rcParams

#plt.figure(figsize=(12, 12))

rcParams['figure.figsize'] = 14, 8

plt.subplot(222)

plt.scatter(X_test[:, 0], X_test[:, 1], c=knn_predicted_test_labels)

plt.title(" Number of Blobs")


#scoring knn
```

```
knn_accuracy_score  = accuracy_score(y_test,knn_predicted_test_labels)

knn_precison_score  = precision_score(y_test,knn_predicted_test_labels)

knn_recall_score    = recall_score(y_test,knn_predicted_test_labels)

knn_f1_score        = f1_score(y_test,knn_predicted_test_labels)

knn_MCC             = matthews_corrcoef(y_test,knn_predicted_test_labels)


#printing

print("")

print("K-Nearest Neighbours + Naive Bayes")

print("Scores")

print("Accuracy -->",knn_accuracy_score)

print("Precison -->",knn_precison_score)

print("Recall -->",knn_recall_score)

print("F1 -->",knn_f1_score)

print("MCC -->",knn_MCC)

print(classification_report(y_test,knn_predicted_test_labels))


import seaborn as sns

LABELS = ['Normal', 'Fraud']

conf_matrix = confusion_matrix(y_test, knn_predicted_test_labels)

plt.figure(figsize=(12, 12))

sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True,
fmt="d");

plt.title("Confusion matrix")
```

```
plt.ylabel('True class')

plt.xlabel('Predicted class')

plt.show()
```

Proposed Methodology

```
dataset = pd.read_csv("/content/drive/MyDrive/Mtech/creditcard.csv")


print(dataset["Class"].value_counts())


minority_ids = dataset[dataset["Class"] == 1].index


majority_ids = dataset[dataset["Class"] == 0].index


sampled_ids = random.sample(list(majority_ids), len(minority_ids))


dataset = pd.concat([dataset.loc[minority_ids],dataset.loc[sampled_ids]])


print(dataset["Class"].value_counts())


All = dataset.shape[0]

fraud = dataset[dataset['Class'] == 1]

nonFraud = dataset[dataset['Class'] == 0]
```

```
x = len(fraud)/All

y = len(nonFraud)/All


print('frauds :',x*100,'%')

print('non frauds :',y*100,'%')


labels = ['non frauds','fraud']

classes = pd.value_counts(dataset['Class'], sort = True)

classes.plot(kind = 'bar', rot=0)

plt.title("Transaction class distribution")

plt.xticks(range(2), labels)

plt.xlabel("Class")

plt.ylabel("Frequency")


# Standardizing the features

dataset['Vamount'] = StandardScaler().fit_transform(dataset['Amount'].values.re-
shape(-1,1))

dataset['Vtime'] = StandardScaler().fit_transform(dataset['Time'].values.reshape(-
1,1))


dataset = dataset.drop(['Time','Amount'], axis = 1)

dataset.head()
```

```
X = dataset.drop(['Class'], axis = 1)

y = dataset['Class']


pca = PCA(n_components=2)

principalComponents = pca.fit_transform(X.values)

principalDf = pd.DataFrame(data = principalComponents

        , columns = ['principal component 1', 'principal component 2'])


features = dataset.drop(['Class'], axis = 1)

labels = pd.DataFrame(dataset['Class'])


feature_array = features.values

label_array = labels.values


# splitting the faeture array and label array keeping 80% for the trainnig sets

X_train,X_test,y_train,y_test = train_test_split(feature_array,label_ar-
ray,test_size=0.20)


# normalize: Scale input vectors individually to unit norm (vector length).

X_train = normalize(X_train)

X_test=normalize(X_test)



neighbours = np.arange(1,25)
```

```
train_accuracy =np.empty(len(neighbours))

test_accuracy = np.empty(len(neighbours))


for i,k in enumerate(neighbours):
    #Setup a knn classifier with k neighbors
    knn=KNeighborsClassifier(n_neighbors=k,algorithm="kd_tree",n_jobs=-1)

    #Fit the model
    knn.fit(X_train,y_train.ravel())

    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(X_train, y_train.ravel())

    #Compute accuracy on the test set
    test_accuracy[i] = knn.score(X_test, y_test.ravel())

plt.title('k-NN Varying number of neighbors')
plt.plot(neighbours, test_accuracy, label='Testing Accuracy')
plt.plot(neighbours, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```

```
idx = np.where(test_accuracy == max(test_accuracy))

x = neighbours[idx]


#k_nearest_neighbours_classification

knn=KNeighborsClassifier(n_neighbors=x[0],algorithm="kd_tree",n_jobs=-1)

knn.fit(X_train,y_train.ravel())


idx = np.where(test_accuracy == max(test_accuracy))

x = neighbours[idx]



filename = 'finalized_model.sav'

joblib.dump(knn, filename)


# load the model from disk

knn = joblib.load(filename)


# predicting labels for testing set

knn_predicted_test_labels=knn.predict(X_test)


from pylab import rcParams

#plt.figure(figsize=(12, 12))
```

```
rcParams['figure.figsize'] = 14, 8

plt.subplot(222)

plt.scatter(X_test[:, 0], X_test[:, 1], c=knn_predicted_test_labels)

plt.title(" Number of Blobs")


#scoring knn

knn_accuracy_score  = accuracy_score(y_test,knn_predicted_test_labels)

knn_precison_score  = precision_score(y_test,knn_predicted_test_labels)

knn_recall_score    = recall_score(y_test,knn_predicted_test_labels)

knn_f1_score        = f1_score(y_test,knn_predicted_test_labels)

knn_MCC             = matthews_corrcoef(y_test,knn_predicted_test_labels)


#printing

print("")

print("K-Nearest Neighbours")

print("Scores")

print("Accuracy -->",knn_accuracy_score)

print("Precison -->",knn_precison_score)

print("Recall -->",knn_recall_score)

print("F1 -->",knn_f1_score)

print("MCC -->",knn_MCC)

print(classification_report(y_test,knn_predicted_test_labels))
```