21AIE205

PYTHON FOR MACHINE LEARNING

CREDIT CARD FRAUD DETECTION USING KNN AND NAIVE BAYES ALGORITHMS

PROJECT REPORT

Submitted to:

MS GANGA GOWRI.B

BATCH-A GROUP-15

1. SYKAM SUMANJALI (CB.EN.U4AIE21068)
2. SUDA HARI PRIYA (CB.EN.U4AIE21067)
3. SAI SRI DARSINI H (CB.EN.U4AIE21055)
4. KURAGAYALA DEVI PRASAD (CB.EN.U4AIE21026)

ACKNOWLEDGEMENTS

We would like to thank Ms. Ganga Gowri.B, our professor-in-charge, for her support and guidance in classes and for helping us in completing our project on the topic. It was a great learning experience.

GROUP-15.

TABLE OF CONTENTS:

INTRODUCTION

The majority of financial institutions have increased their availability of business services to the general public through internet banking in the twenty-first century. In today's financially competitive culture, e-payment systems are essential. They have made it quite simple to buy products and services. Customers of financial institutions are frequently given cards that make their lives easier when they shop without carrying cash. In addition to debit cards, credit cards are advantageous to customers since they shield them from potentially damaged, misplaced, or even stolen products. Before making any purchases with their credit or debit card, customers must confirm the transaction with the retailer.

Despite the many advantages that credit cards offer users, they are also linked to issues like fraud and security. Banks and other financial organizations are said to be struggling with the theft of credit and debit cards. It happens when unauthorized persons use credit cards fraudulently to get money or property. Theft of credit card information can happen through unprotected internet platforms and web pages. Additionally, they can be acquired through identity theft methods. Users' credit and debit card details are vulnerable to unauthorized access by fraudsters, who may do so without their knowledge or agreement.

MOTIVATION OF THE PROJECT

Machine learning is competent in distinguishing between real and fraudulent transactions. The difficulty in communicating ideas on fraud detection is one of the major problems with detection methods. There are a lot of different types of credit card transactions. Based on their regions and currencies, customers utilize credit cards for a variety of things, demonstrating the vast variety of fraudulent transactions. This issue has inspired us to come up with a solution that could help identify fraudulent transactions regardless of location.

The dataset used here has been collected and analyzed during a research collaboration between Worldline and the Machine Learning Group of ULB (Université Libre de Bruxelles) on big data mining and fraud detection. The dataset has 492 frauds out of 284,807 transactions that took place over the course of two days. It only has numeric input variables that have undergone PCA transformation. Unfortunately, the dataset is unable to offer the original characteristics and further context for the data due to confidentiality concerns. The major components derived from PCA are features V1, V2,..., V28. The only features that have not been changed with PCA are "Time" and "Amount." The seconds that passed between each transaction and the dataset's initial transaction are listed in the feature "Time." The transaction amount is represented by the feature "Amount," which may be utilized for

example-dependent, cost-sensitive learning. The response variable, feature "Class," has a value of 1 in cases of fraud and 0 in all other cases.

We use two algorithms, K-Nearest Neighbours and Naïve Bayes classification algorithms to detect which transactions are fraudulent and which transactions are not fraudulent.

DATA PRE-PROCESSING

First, import the necessary libraries needed for the dataset. Then load the dataset and display it.

Importing the libraries

```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, recall_score, precision_score, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
import warnings
warnings.simplefilter('ignore')
from collections import Counter
```

Loading the dataset

```
df = pd.read_csv(r"C:\Users\Darsini\Downloads\creditcard_csv.csv")
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

5 rows × 31 columns

Then show the information about the dataset.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

Since the algorithm is large, we are decreasing the dataset size to 10,000 rows.

```
df = df.head(10000)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Time    10000 non-null  float64
 1   V1      10000 non-null  float64
 2   V2      10000 non-null  float64
 3   V3      10000 non-null  float64
 4   V4      10000 non-null  float64
 5   V5      10000 non-null  float64
 6   V6      10000 non-null  float64
 7   V7      10000 non-null  float64
 8   V8      10000 non-null  float64
 9   V9      10000 non-null  float64
 10  V10     10000 non-null  float64
 11  V11     10000 non-null  float64
 12  V12     10000 non-null  float64
 13  V13     10000 non-null  float64
 14  V14     10000 non-null  float64
 15  V15     10000 non-null  float64
 16  V16     10000 non-null  float64
 17  V17     10000 non-null  float64
 18  V18     10000 non-null  float64
 19  V19     10000 non-null  float64
 20  V20     10000 non-null  float64
 21  V21     10000 non-null  float64
 22  V22     10000 non-null  float64
 23  V23     10000 non-null  float64
 24  V24     10000 non-null  float64
 25  V25     10000 non-null  float64
 26  V26     10000 non-null  float64
 27  V27     10000 non-null  float64
 28  V28     10000 non-null  float64
 29  Amount  10000 non-null  float64
 30  Class   10000 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 2.4 MB
```

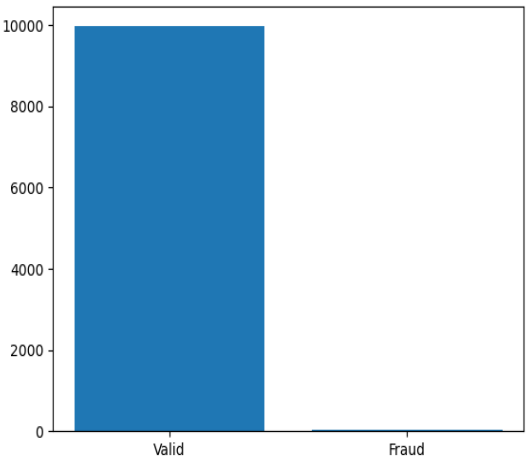
Then the “Time” feature is dropped because it does not provide much information about if the transaction was fraudulent or not, as it lists the seconds that passed between each transaction and the dataset's initial transaction.

```
df = df.drop(['Time'], axis=1)
```

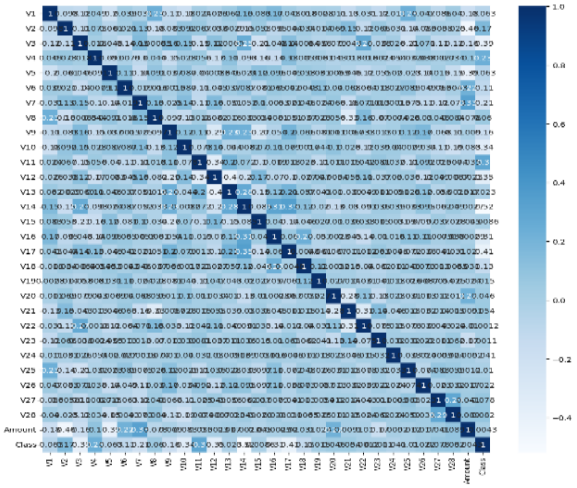
A graph for the number of valid and fraudulent transactions is shown. And, the correlation matrix between all the variables.

```
plt.bar(['Valid','Fraud'],list(df['Class'].value_counts()))
print("Fraudulent transactions: ", end='')
frauds= df['Class'].value_counts()[1]/sum(df['Class'].value_counts())
print(round(frauds*100,2), end='%')
plt.show()
```

```
Fraudulent transactions: 0.38%
```




```
fig=plt.figure(figsize= (12, 12))
sns.heatmap(df.corr(), annot = True,cmap='Blues')
plt.show()
```



FEATURE ENGINEERING

Check for missing values and categorical variables. And find that are no null values and categorical variables.

Check for missing value

```
# check for null values
df.isnull().sum()
```

```
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

Check for categorical variables

```
# find categorical variables
categorical = [var for var in df.columns if df[var].dtype=='O'] # Saving CATEGORICAL Variables
print('There are {} categorical variables\n'.format(len(categorical)))
print('The categorical variables are :', categorical)
```

```
There are 0 categorical variables

The categorical variables are : []
```

Now create a matrix of features in our dataset (X) and create a dependent vector or the target labels (Y) with their respective observations. To read the columns, use ‘iloc’ of pandas.

Here we take all the features, that is, from V1 to Amount in X, and store it as a matrix using ‘.values’. In Y, the target values, that is ‘Class’ column is taken. Then print the unique values in Y.

```
# dividing the x and the y from the dataset
x = df.iloc[:,:-1]
y = df['Class']
# getting just the values for the sake of processing
# (its a numpy array with no columns)
X = x.values
Y = y.values
print(np.unique(y))
```

```
[0 1]
```

Split the data into training sets and testing tests using the train_test_split function. This dataset that we initially feed into a machine learning algorithm to “train” the algorithm, is typically called the training dataset. The test dataset is used as an input to the model after the model has been built to “test” that the model works as needed. The test size given here is 0.45, meaning 45% of the whole dataset is used for testing and then the remaining 55% is used for training.

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.45, random_state=100)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(5500, 29)
(4500, 29)
(5500,)
(4500,)
```

METHODS USED

1. K-NEAREST NEIGHBORS:

KNN is a supervised machine learning algorithm. KNN predicts the values based on the distances of each test data point from the training dataset. KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much like the new data.

How this algorithm works is first we need an optimal value for K to start, then calculate the distance of each data point in the test set with each point in the training set. Then, sort the calculated distances along with the corresponding target values

from training data in ascending order. From these sorted values, select the K top values. We took K=3.

i. The ‘distance_ecu’ function calculates the Euclidean distance between a test data point X_test and every data point in the training data X_train. The function loops over the rows of X_train and calculates the distance between the test data point and the current training data point by summing the square of the differences between their corresponding features. The Euclidean distance is then calculated as the square root of the sum of these differences. The distances between the test data point and each training data point are stored in a data frame and returned by the function.
ii. The ‘nearest_neighbors’ function sorts the distances between the test point and each point in the training data in ascending order using the ‘.sort_values’ function along the row and returns the first K closest points, stored in a data frame.
iii. The ‘voting’ function then computes the prediction based on the majority voting of the labels of the K nearest neighbors. The prediction is made by using the Counter object to get the labels with the K nearest neighbors. The most common label among the K nearest neighbors is then selected as the prediction, and the function returns the prediction as the output.
iv. The function ‘KNN_from_scratch’ takes each test point in X_test, calculates the Euclidean distances, selects the K nearest neighbors, predicts the label, and returns a list of predictions, one for each test point.

2. NAÏVE BAYES:

The Naïve Bayes algorithm is a supervised learning algorithm, which is based on the Bayes theorem and used for solving classification problems. It is one of the simple and most effective Classification algorithms which helps in building fast machine-learning models that can make quick predictions.

Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.

- The formula for Bayes' theorem is given as $P(A|B) = P(B|A)*P(A) / P(B)$ where, $P(A|B)$ is Posterior probability: the probability of the dependent variable (target variable) A on the independent event (feature variables) B,
- $P(B|A)$ is Likelihood probability: the probability of the feature variables given that the probability of a target variable is true,
- $P(A)$ is Prior Probability: the probability of the target variable before observing the feature variables and $P(B)$ is Marginal Probability: the probability of feature variables.

How this algorithm works is:

i. 'fit’: This method takes in the feature matrix X and target vector y as input and fits the Naive Bayes classifier to the data by calculating the mean, variance, and prior probabilities for each class.

ii. ‘predict’: This method takes in a feature matrix X and returns the predicted class labels for each row.

iii. ‘_predict’: This method is a helper function that takes in a single feature vector x and returns the predicted class label for that instance. The np.argmax function returns the index of the highest value in the posteriors array, which represents the class with the highest probability. The class label for this class is then obtained by indexing self._classes with the index returned by np.argmax.

iv. ‘_pdf’: This method calculates the probability density function (PDF) for a given class and feature vector, which is used in the calculation of posterior probabilities.

Overall, this implementation uses the Gaussian Naive Bayes approach, which assumes that the features are normally distributed and calculates the probabilities using the Gaussian PDF.

RESULTS WITH INFERENCES

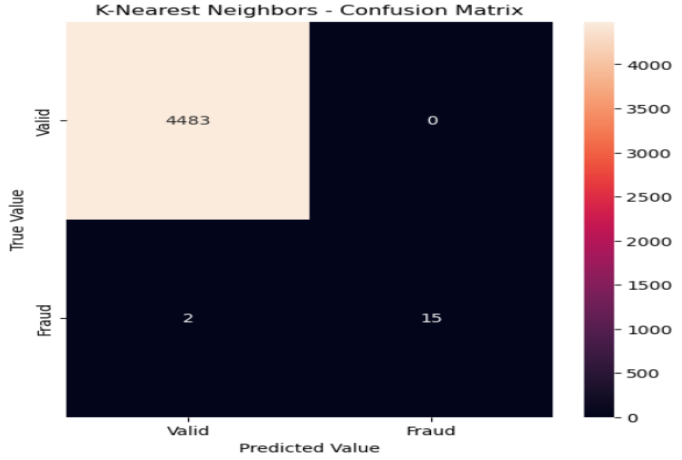
To get the performance of the model, Accuracy, Precision, Recall, F1-Score, and Confusion matrix are used as the evaluation metrics.

I. For KNN algorithm:

```
print("Accuracy score: ", accuracy_score(y_true=y_test, y_pred=y_pred_KNN))
print("Precision: ", precision_score(y_test, y_pred_KNN, average='macro'))
print("Recall: ", recall_score(y_test, y_pred_KNN, average='macro'))
print("F1-score: ", f1_score(y_test, y_pred_KNN, average='macro'))
```

```
Accuracy score:  0.9995555555555555
Precision:  0.9997770345596433
Recall:  0.9411764705882353
F1-score:  0.9686384924174843
```

```
cm_knn = confusion_matrix(y_test, y_pred_KNN)
labels= ['Valid', 'Fraud']
plt.figure(figsize=(6, 6))
sns.heatmap(cm_knn, xticklabels= labels, yticklabels= labels, annot=True, fmt="d")
plt.title("K-Nearest Neighbors - Confusion Matrix")
plt.ylabel('True Value')
plt.xlabel('Predicted Value')
plt.show()
```



For the KNN algorithm, we got an accuracy score of 99.95%.

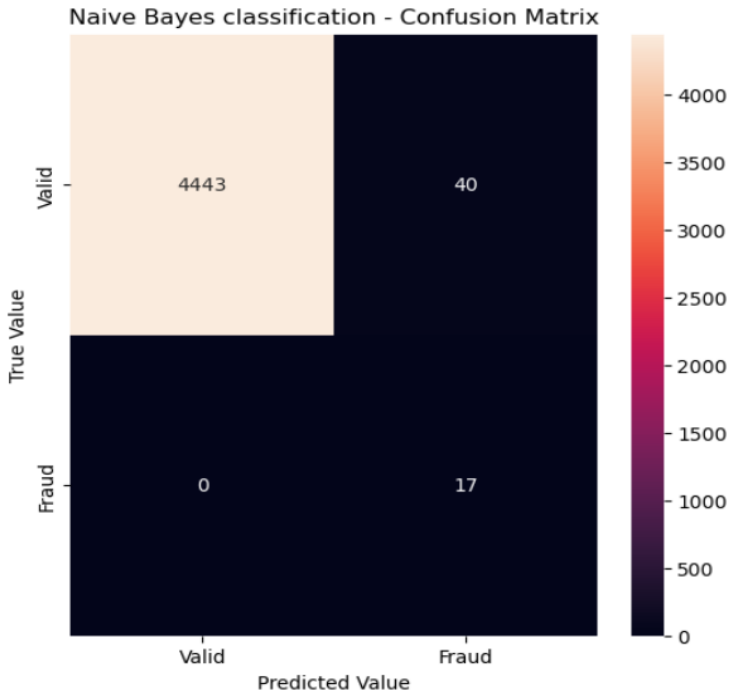
II. For Naïve Bayes algorithm:

```
nb = NaiveBayes()
nb.fit(X_train, y_train)
y_pred_NB = nb.predict(X_test)

print("Naive Bayes classification accuracy", accuracy_score(y_test, y_pred_NB))
print("Precision: ", precision_score(y_test, y_pred_NB, average='macro'))
print("Recall: ", recall_score(y_test, y_pred_NB, average='macro'))
print("F1-score: ", f1_score(y_test, y_pred_NB, average='macro'))
```

```
Naive Bayes classification accuracy 0.9911111111111112
Precision:  0.6491228070175439
Recall:  0.9955387017622128
F1-score:  0.7274890844238817
```

```
cm_nb = confusion_matrix(y_test, y_pred_NB)
labels= ['Valid', 'Fraud']
plt.figure(figsize=(6, 6))
sns.heatmap(cm_nb, xticklabels= labels, yticklabels= labels, annot=True, fmt="d")
plt.title("Naive Bayes classification - Confusion Matrix")
plt.ylabel('True Value')
plt.xlabel('Predicted Value')
plt.show()
```



For the Naïve Bayes algorithm, we got an accuracy of 99.11%.

Credit card fraud detection differs from other machine learning issues in that the target class distribution is not uniform. It is often referred to as the imbalanced data issue or the class imbalance problem.

Not everything depends on accuracy. Accuracy is not a realistic performance indicator when dealing with extremely unbalanced data. Because you can get a greater accuracy by doing nothing more than assuming that everything belongs to the major class than you can by creating a predictive model. Since fraudulent transactions may be highly expensive and false alarms result in someone's transaction being stopped, a credit card

firm wants to keep an eye on fraud as much as possible (minimize false negatives and reduce false positives). Therefore, the credit card corporation seeks to maximize recall. F1-score takes into account a balance between recall and accuracy.

REFERENCES

1. 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
2. https://github.com/aswintechguy/Machine-Learning-Projects/tree/master/Credit%20Card%20Fraud%20Detection%20-%20Classification
3. https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
4. https://dataaspirant.com/credit-card-fraud-detection-classification-algorithms-python/
5. https://www.javatpoint.com/machine-learning-naive-bayes-classifier
6. https://towardsdatascience.com/create-your-own-k-nearest-neighbors-algorithm-in-python-eb7093fc6339
7. https://medium.com/@rangavamsi5/na%C3%AFve-bayes-algorithm-implementation-from-scratch-in-python-7b2cc39268b9
8. https://www.kdnuggets.com/2020/06/naive-bayes-algorithm-everything.html
9. https://medium.com/analytics-vidhya/credit-card-fraud-detection-in-depth-study-evaluating-the-classification-model-a3680a5a5897