

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

**J COMPONENT REPORT**

**PROGRAMME:** BTECH CSE CORE

**COURSE TITLE:** MACHINE LEARNING

**COURSE CODE:** CSE4020

**SLOT:** D2

**TITLE**

CREDIT CARD FRAUD DETECTION

**TEAM MEMBERS**

HRITHIK UMESH – 20BCE1319

ANIRUDH VANNARATH- 20BCE1718

**ACKNOWLEDGEMENT**

We express our earnest gratitude and utmost appreciation to Dr Tulasi Prasad Sariki, Associate Professor, School of Computer Science and Engineering, VIT Chennai, who has been a constant source of encouragement and whose guidance has been invaluable for the duration of our project. We would also like to extend our thanks to the entire faculty of the school for their unwavering support and for imparting their extensive knowledge and expertise to us during the course of our project. We are also deeply grateful to our parents, family members, and friends who have patiently supported us throughout our project journey and provided us with the opportunity to pursue our academic aspirations in such a renowned institution. Their unwavering support and encouragement have been instrumental in our success.

**ABSTRACT**

The project aims to develop a machine learning model that can detect fraudulent credit card transactions with high accuracy. The model will be trained on a dataset of credit card transactions that includes both fraudulent and non-fraudulent transactions. The dataset will be pre-processed to remove any irrelevant features and to balance the number of fraudulent and non-fraudulent transactions. The model will be developed using a supervised learning algorithm such as logistic regression or decision trees. The performance of the model will be evaluated using metrics such as precision, recall, and F1 score. Once the model is developed and tested, it can be integrated into a real-time credit card fraud detection system to help prevent fraudulent transactions.

**INTRODUCTION**

Credit card fraud is a serious problem that affects millions of people around the world. Fraudulent transactions can result in significant financial losses for both individuals and financial institutions. To combat this problem, many financial institutions use machine learning algorithms to detect fraudulent transactions in real-time. These algorithms are trained on large datasets of credit card transactions that include both fraudulent and non-fraudulent transactions. By analysing patterns in the data, these algorithms can identify transactions that are likely to be fraudulent and flag them for further investigation.

The process of developing a machine learning model for credit card fraud detection involves several steps. The first step is to collect a large dataset of credit card transactions that includes both fraudulent and non-fraudulent transactions. This dataset must be pre-processed to remove any irrelevant features and to balance the number of fraudulent and non-fraudulent transactions.

Once the dataset has been pre-processed, the next step is to develop a machine learning model using a supervised learning algorithm such as logistic regression or decision trees. The model is trained on the pre-processed dataset and evaluated using metrics such as precision, recall, and F1 score. The goal is to develop a model that can accurately detect fraudulent credit card transactions with high precision and recall.

There are several challenges associated with developing a machine learning model for credit card fraud detection. One of the biggest challenges is dealing with imbalanced datasets. Since fraudulent transactions are relatively rare compared to non-fraudulent transactions, the dataset may be heavily skewed towards non-fraudulent transactions. This can lead to poor performance of the machine learning model since it may not be able to accurately detect fraudulent transactions.

Another challenge is dealing with evolving fraud patterns. Fraudsters are constantly coming up with new ways to commit credit card fraud, which means that the machine learning model must be able to adapt to these new patterns in order to remain effective.

Despite these challenges, machine learning algorithms have proven to be highly effective at detecting credit card fraud in real-time. By using these algorithms, financial institutions can significantly reduce the risk of financial losses due to fraudulent transactions.

**REQUIREMENT SPECIFICATION**

* About the dataset, it 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.
* It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. 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-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.
* Python and its libraires like Matplotlib, NumPy, Pandas, Sklearn including Standard Scaler and split, etc. are used for this project.
* Google collab IDE environment for EDA and Model Training

**PROPOSED METHODOLOGY**

In this project, the challenge is to recognize fraudulent credit card transactions so that the customers of credit card companies are not charged for items that they did not purchase. We make use of various models like Decision Tree, Logistic Regression, KNN, Support Vector machines and Neural Networks and compare the prediction accuracy of these models using Confusion matrix.

The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data. It can only be determined if the true values for test data are known. The matrix itself can be easily understood, but the related terminologies may be confusing. Since it shows the errors in the model performance in the form of a matrix, hence also known as an error matrix.

It evaluates the performance of the classification models, when they make predictions on test data, and tells how good our classification model is. It not only tells the error made by the classifiers but also the type of errors such as it is either type-I or type-II error. With the help of the confusion matrix, we can calculate the different parameters for the model, such as accuracy, precision, etc.

**IMPLEMENTATION**

**Importing libraries**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.metrics import f1\_score

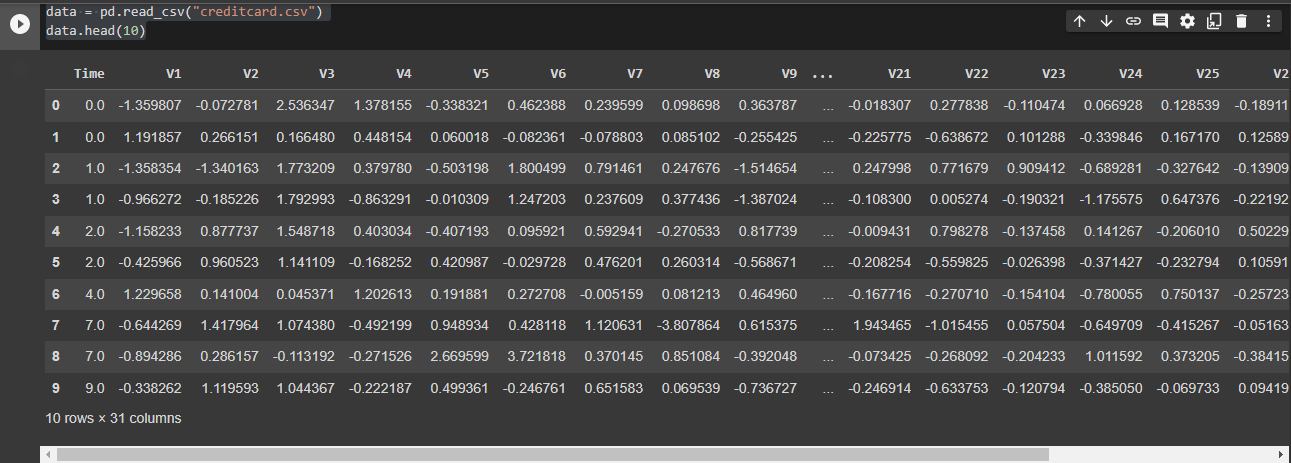
from sklearn.metrics import accuracy\_score, confusion\_matrix, mean\_squared\_error, f1\_score

from sklearn import metrics

**Loading the dataset**

data = pd.read\_csv("creditcard.csv")

data.head(10)



**Processing the data**

Total\_transactions = len(data)

normal = len(data[data.Class == 0])

fraudulent = len(data[data.Class == 1])

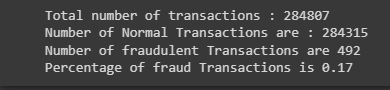
fraud\_percentage = round(fraudulent/normal\*100, 2)

print(f"Total number of transactions : {Total\_transactions}")

print(f'Number of Normal Transactions are : {normal}')

print(f'Number of fraudulent Transactions are {fraudulent}')

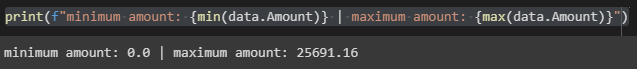
print(f'Percentage of fraud Transactions is {fraud\_percentage}')



After using data.info() we find out that there is no null values in the dataset.

#### **In the dataset all the features are transformed using PCA except Amount**

print(f"minimum amount: {min(data.Amount)} | maximum amount: {max(data.Amount)}")



**Since there is a huge difference between minimum and maximum value of the amount, we scale this variable**

sc = StandardScaler()

amount = data['Amount'].values

data['Amount'] = sc.fit\_transform(amount.reshape(-1, 1))

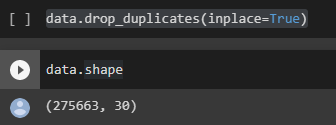
**We drop the time variable**

data.drop(['Time'], axis=1, inplace=True)

#### **Dropping duplicate values**

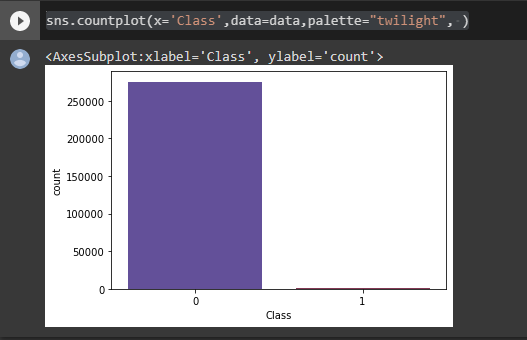
data.drop\_duplicates(inplace=True)

data.shape



**9144 duplicate values have been dropped.**

sns.countplot(x='Class',data=data,palette="twilight", )



**From the recorded transactions, a very small portion is fraudulent, which suggests that the data is highly unbalanced.**

**Correlation between variables**

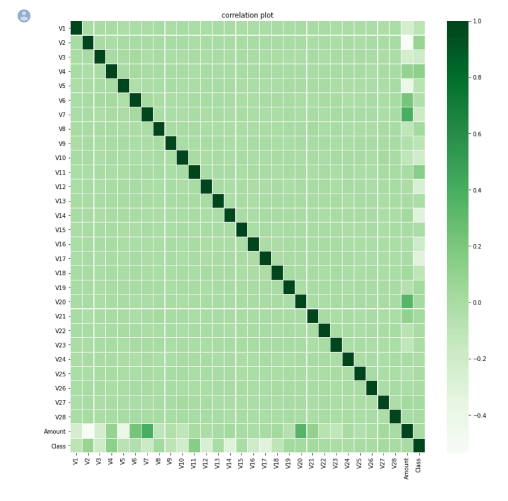
plt.figure(figsize = (14,14))

plt.title('correlation plot')

corr = data.corr()

sns.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns,linewidths=.1,cmap="Greens")

plt.show()



**Function to plot confusion matrix**

def display\_confusion\_matrix(y\_test,y\_pred):

matrix = metrics.confusion\_matrix(y\_test,y\_pred)

matrixDisplay = metrics.ConfusionMatrixDisplay(confusion\_matrix = matrix, display\_labels = [False, True])

matrixDisplay.plot()

plt.show()

**Function to get metrics of the predicted data**

def show\_metrics(ypred, ytest, model\_name):

accuracy = accuracy\_score(ypred, ytest)

mse = mean\_squared\_error(ypred, ytest)

rmse = np.sqrt(mse)

f1 = f1\_score(ypred, ytest)

dict = {'details':['model name', 'accuracy', 'mean squared error', 'root mean squared error', 'f1 score'], 'score':[model\_name, accuracy, mse, rmse, f1]}

metrics\_data = pd.DataFrame(dict)

display(metrics\_data)

**Train and test split**

X = data.drop('Class', axis = 1).values

y = data['Class'].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 1)

**Model building**

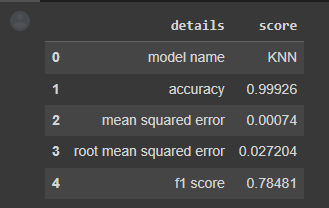
n = 8

model\_1 = KNeighborsClassifier(n\_neighbors = n)

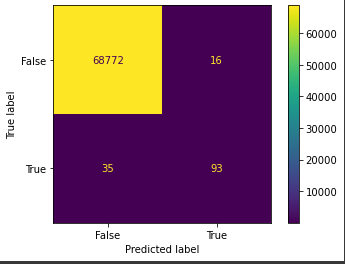
model\_1.fit(X\_train, y\_train)

y\_pred\_1 = model\_1.predict(X\_test)

show\_metrics(y\_pred\_1, y\_test, 'KNN')



display\_confusion\_matrix(y\_test,y\_pred\_1)



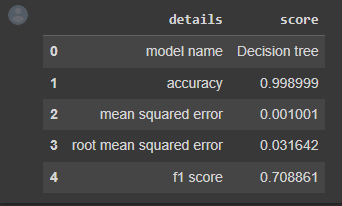
### **Decision tree**

model\_2 = DecisionTreeClassifier(random\_state = 0)

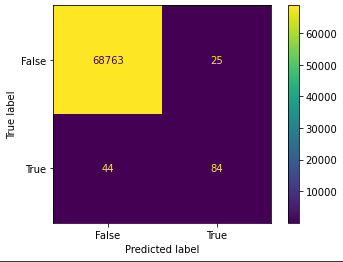
model\_2.fit(X\_train,y\_train)

y\_pred\_2 = model\_2.predict(X\_test)

show\_metrics(y\_pred\_2, y\_test, 'Decision tree')



display\_confusion\_matrix(y\_test,y\_pred\_2)



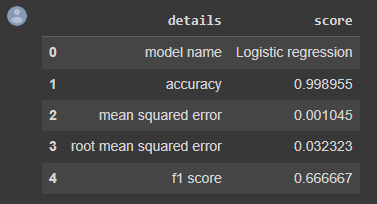
**Logistic Regression**

model\_3 = LogisticRegression()

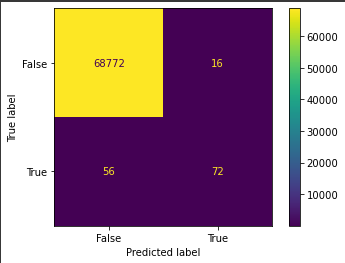
model\_3.fit(X\_train, y\_train)

y\_pred\_3 = model\_3.predict(X\_test)

show\_metrics(y\_pred\_3, y\_test, 'Logistic regression')



display\_confusion\_matrix(y\_test, y\_pred\_3)



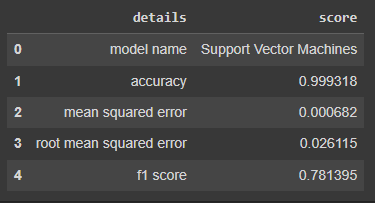
**Support Vector Machines**

model\_4 = SVC()

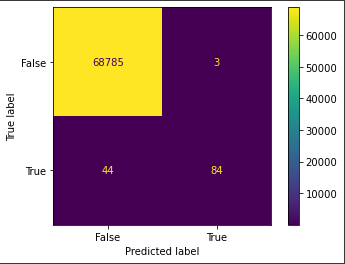
model\_4.fit(X\_train, y\_train)

y\_pred\_4 = model\_4.predict(X\_test)

show\_metrics(y\_pred\_4, y\_test, 'Support Vector Machines')



display\_confusion\_matrix(y\_test, y\_pred\_4)



**Neural Networks**

from sklearn.neural\_network import MLPClassifier

model\_5 = MLPClassifier(hidden\_layer\_sizes = (6,5),

activation='relu',

solver='adam',

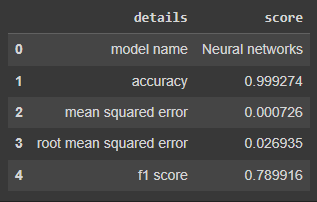
verbose=False,

random\_state=45)

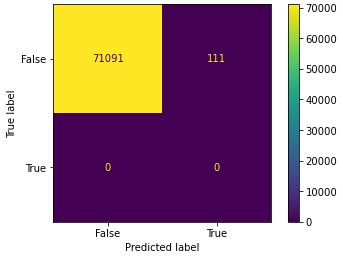
model\_5.fit(X\_train, y\_train)

y\_pred\_5 = model\_5.predict(X\_test)

show\_metrics(y\_pred\_5, y\_test, 'Neural networks')



display\_confusion\_matrix(y\_pred\_5, y\_test)



**Model comparison**

def compare\_models(y\_pred, y\_test):

models = ['KNN', 'Decision tree', 'Logistic Regression', 'Support vector machines', 'Neural networks']

accuracy = []

mse = []

rmse = []

f1 = []

for i in range(len(models)):

accuracy.append(accuracy\_score(y\_pred[i], y\_test))

mse.append(mean\_squared\_error(y\_pred[i], y\_test))

rmse.append(np.sqrt(mean\_squared\_error(y\_pred[i], y\_test)))

f1.append(f1\_score(y\_pred[i], y\_test))

dict = {'model':models, 'accuracy':accuracy, 'mean squared error':mse, 'root mean squared error': rmse, 'f1 score':f1}

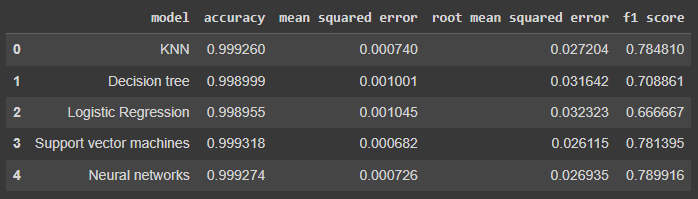
compare\_df = pd.DataFrame(dict)

return compare\_df

predicted\_values = [y\_pred\_1, y\_pred\_2, y\_pred\_3, y\_pred\_4, y\_pred\_5]

compare\_df = compare\_models(predicted\_values, y\_test)

display(compare\_df)



**OBSERVATIONS AND RESULTS**

From the above comparison data frame, in terms of accuracy, mean squared error, root mean squared error and f1 score, Neural networks yields a better prediction when compared to the other models. So, we choose this model for further study to develop a model that can accurately predict the probability of a fraudulent credit card transaction.