Credit Card Fraud Detection.ipynb

import numpy as np

import pandas as pd

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

#NUMPY- NumPy can be used to perform a wide variety of mathematical operations on arrays

#Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.

2.helps us to make data frame

#DATA FRAME - structured table

#train\_test\_split- allows us to split our data into training data and test data

#Logistic regression model- Logistic regression transforms the continuous output of a linear regression model into a categorical value (0 or 1) using a sigmoid function, which maps any input to a value between 0 and 1.

This is done to model binary classification problems.

#accuracy\_score- tells you how often your model's predictions match the actual outcomes

# loading the dataset to a Pandas DataFrame

credit\_card\_data = pd.read\_csv('/content/credit\_data.csv')

-load our dataset to a panda data frame

-v1 to v28 are vertical transactions (credit card details are sensitive details.)

- Dataset provider converted all the features through principal component analysis(PCA) and convert features into numerical value.

-we will use these numerical values for our analysis and prediction.

- CLASS means whether the transaction is LEGIT or FRAUDULENT transaction. (0 and 1 respectively.)

-currency is in DOLLAR and Time is in seconds

# dataset informations

credit\_card\_data.info()

-dataset informations

-tell entries , data type and how many values are present

credit\_card\_data.isnull().sum()

-checking the number of missing values in each column

credit\_card\_data['Class'].value\_counts()

- distribution of legit transactions & fraudulent transactions

- 0 --> Normal Transaction (284315)

- 1 --> fraudulent transaction (492)

-This Dataset is highly unblanced

- We have two classes (two target variables ) here , because in this case more than 99 percent data is in a particular class.

-We cannot feed this data to our machine learning model because if we train machine learning model with this data , then it cannot recognize fraudulent transactions because we have very less data points .

-then processing comes into play

legit = credit\_card\_data[credit\_card\_data.Class == 0]

fraud = credit\_card\_data[credit\_card\_data.Class == 1]

separating the data for analysis

print(legit.shape)

print(fraud.shape)

(284315, 31)

(492, 31)

* 31 are columns.
* 492 are fraudulent transactions.
* 284315 are legit transactions.

legit.Amount.describe()

-statistical measures of the data

fraud.Amount.describe()

-mean is bigger in fraudulent.

credit\_card\_data.groupby('Class').mean()

-compare the values for both transactions

Legit\_sample = legit.sample(n=492)

-Under-Sampling

Build a sample dataset containing similar distribution of normal transactions and Fraudulent Transactions

Number of Fraudulent Transactions --> 492

new\_dataset = pd.concat([legit\_sample, fraud], axis=0)

-Concatenating two DataFrames

-axis =0 means row wise data

new\_dataset.head()

new\_dataset.tail()

new\_dataset['Class'].value\_counts()

new\_dataset.groupby('Class').mean()

Splitting the data into Features & Targets

Splitting -> V1 , V2, and so on

Target -> 0 and 1

X = new\_dataset.drop(columns='Class', axis=1)

Y = new\_dataset['Class']

print(X)

print(Y)

Split the data into Training data & Testing Data

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, stratify=Y, random\_state=2)

-features of training data store in X\_train (80 % data)

-All labels of corresponding data store in Y\_train (20% )

print(X.shape, X\_train.shape, X\_test.shape)

(984, 30) (787, 30) (197, 30)

Model Training

Logistic Regression

model = LogisticRegression()

# training the Logistic Regression Model with Training Data

model.fit(X\_train, Y\_train)

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, l1\_ratio=None, max\_iter=100,

multi\_class='auto', n\_jobs=None, penalty='l2',

random\_state=None, solver='lbfgs', tol=0.0001, verbose=0,

warm\_start=False)

Model Evaluation

Accuracy Score

# accuracy on training data

X\_train\_prediction = model.predict(X\_train)

training\_data\_accuracy = accuracy\_score(X\_train\_prediction, Y\_train)

print('Accuracy on Training data : ', training\_data\_accuracy)

Accuracy on Training data : 0.9415501905972046

# accuracy on test data

X\_test\_prediction = model.predict(X\_test)

test\_data\_accuracy = accuracy\_score(X\_test\_prediction, Y\_test)

print('Accuracy score on Test Data : ', test\_data\_accuracy)

Accuracy score on Test Data : 0.9390862944162437

NOTE:

Why we have predicted accuracy score of training data ?

* If accuracy score of training data is very different from test data, then it makes our model over fitted or under fitted.

-Let’s say if we get accuracy score of 95% in training data and only 50% in test data ,that means our model is over fitted with training data.

-It means model is overtrain on model data.

-In underfitting , we get very less training data accuracy.