EXPERIMENT 2

PART B

|  |  |
| --- | --- |
| Roll No. C009 | Name: Samarth Borade |
| Class : BTI SEM 10 | Batch : EB1 |
| Date of Experiment: 22/12/23 | Date of Submission: 22/12/23 |
| Grade : |  |

**B.1 Software Code written by student:**

# Samarth Borade

# BTI SEM 10

# C009

# EXP 2

# Aim: Implementing Multilayer Perceptron

****

# Modules used for data handling and linear algebra operations.

import pandas as pd

import numpy as np

# Modules used for data visualization.

import matplotlib.pyplot as plt

import seaborn as sns

sns.set\_style()

# Modules used for preprocessing

from sklearn.preprocessing import OneHotEncoder

# Modules used for Machine Learning models.

from sklearn.linear\_model import Perceptron

from sklearn.neural\_network import MLPClassifier

# Modules used for hyperparameter tuning.

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import RandomizedSearchCV

# Models used for evaluating the model.

from sklearn import metrics

from sklearn.model\_selection import cross\_val\_score

# Suppressing the warnings.

import warnings

warnings.filterwarnings('ignore')

**Dataset 1:**

df = pd.read\_csv("crx.csv", header=None)

headers = df.iloc[0]

df  = pd.DataFrame(df.values[1:], columns=headers)

cat\_cols = []

num\_cols = []

for i in df.columns:

    if df[i].dtype == "O":

        cat\_cols.append(i)

    else:

        num\_cols.append(i)

null\_freq = []

for i in df.columns:

    f = dict(df[i].value\_counts())

    if "?" in f.keys():

        null\_freq.append(f["?"]\*100/len(df))

    else:

        null\_freq.append(0)

pd.Series(dict(zip(df.columns,null\_freq))).plot(kind="bar",

                                                rot=0,

                                                title="Missing Value Frequency",

                                                xlabel="Column Name",

                                                ylabel="Percentage of missing values",

                                                color=["orange","crimson"])

plt.show()

df['a16'].value\_counts().plot(kind="bar",

                           title="Class Distribution",

                           xlabel="Status of Credit Card Approval",

                           ylabel="Frequency of the Status",

                           color=["crimson","orange"],

                           rot=0)

plt.show()

df = df.replace({"?":None})

df = df.dropna()

encoder = OneHotEncoder(sparse=False)

for i in cat\_cols:

    df[i] = encoder.fit\_transform(df[i].values.reshape(-1,1))

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

X\_train,X\_test,y\_train,y\_test = train\_test\_split(df,

    df['a16'],

    test\_size = 0.30,

    train\_size=0.70,

    random\_state = 0

)

X\_train.pop('a16')

X\_test.pop('a16')

****

clf = Perceptron(random\_state=0)

clf = Perceptron(random\_state=0)

y\_pred\_train = clf.predict(X\_train)

y\_pred\_test = clf.predict(X\_test)

df.dtypes.value\_counts().plot(kind="bar",

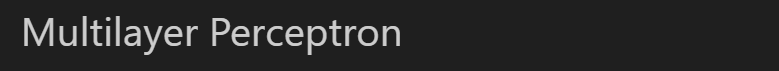
                              title="Types of Data",

                              xlabel="Data Type",

                              ylabel="No.of columns",

                              rot=0,

                              color=["crimson","orange"])

plt.show()

clf = MLPClassifier(random\_state=1, max\_iter=300).fit(X\_train, y\_train)

y\_pred\_train = clf.predict(X\_train)

y\_pred\_test = clf.predict(X\_test)

metrics.accuracy\_score(y\_true=y\_train,y\_pred=y\_pred\_train)

metrics.accuracy\_score(y\_true=y\_test,y\_pred=y\_pred\_test)

**Dataset 2:**

df = pd.read\_csv("cirrhosis.csv", header=None)

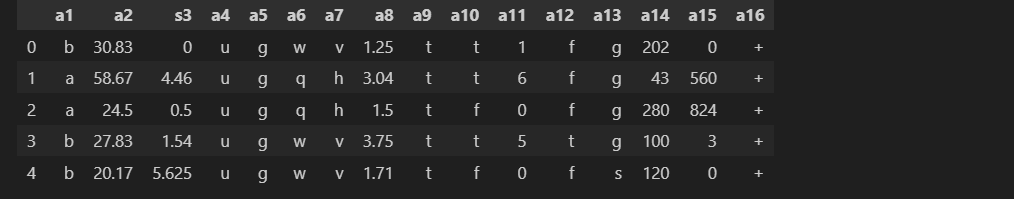
headers = df.iloc[0]

df  = pd.DataFrame(df.values[1:], columns=headers)

**B.2 Input and Output:**

**Input:**

**Dataset 1:**

****

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 690 entries, 0 to 689

Data columns (total 16 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 a1 690 non-null object

1 a2 690 non-null object

2 s3 690 non-null object

3 a4 690 non-null object

4 a5 690 non-null object

5 a6 690 non-null object

6 a7 690 non-null object

7 a8 690 non-null object

8 a9 690 non-null object

9 a10 690 non-null object

10 a11 690 non-null object

11 a12 690 non-null object

12 a13 690 non-null object

13 a14 690 non-null object

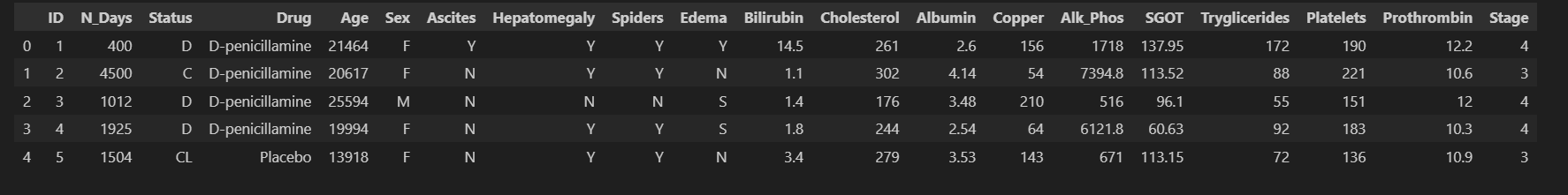
14 a15 690 non-null object

15 a16 690 non-null object

dtypes: object(16)

memory usage: 86.4+ KB

**Dataset 2:**

****

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 418 entries, 0 to 417

Data columns (total 20 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 ID 418 non-null object

1 N\_Days 418 non-null object

2 Status 418 non-null object

3 Drug 312 non-null object

4 Age 418 non-null object

5 Sex 418 non-null object

6 Ascites 312 non-null object

7 Hepatomegaly 312 non-null object

8 Spiders 312 non-null object

9 Edema 418 non-null object

10 Bilirubin 418 non-null object

11 Cholesterol 284 non-null object

12 Albumin 418 non-null object

13 Copper 310 non-null object

14 Alk\_Phos 312 non-null object

15 SGOT 312 non-null object

16 Tryglicerides 282 non-null object

17 Platelets 407 non-null object

18 Prothrombin 416 non-null object

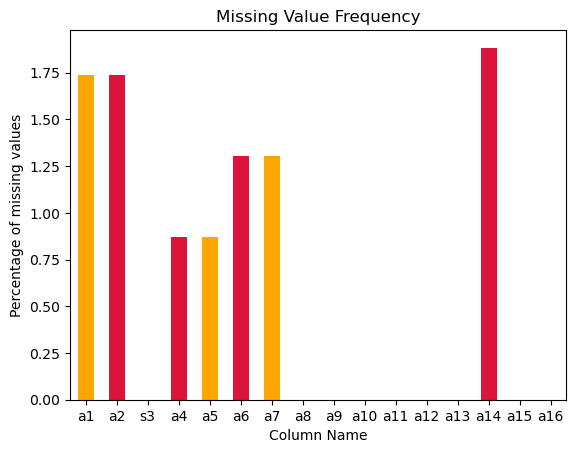
19 Stage 412 non-null object

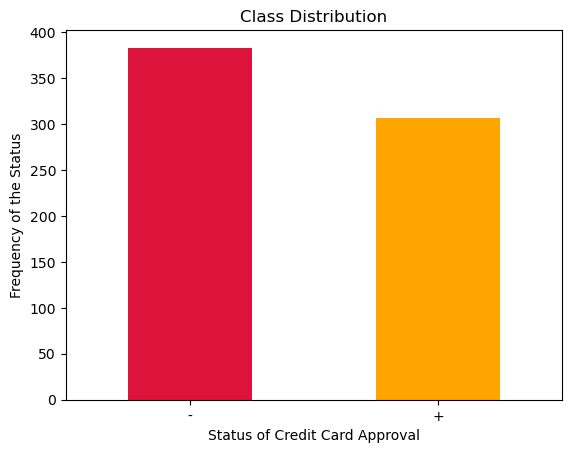
dtypes: object(20)

memory usage: 65.4+ KB

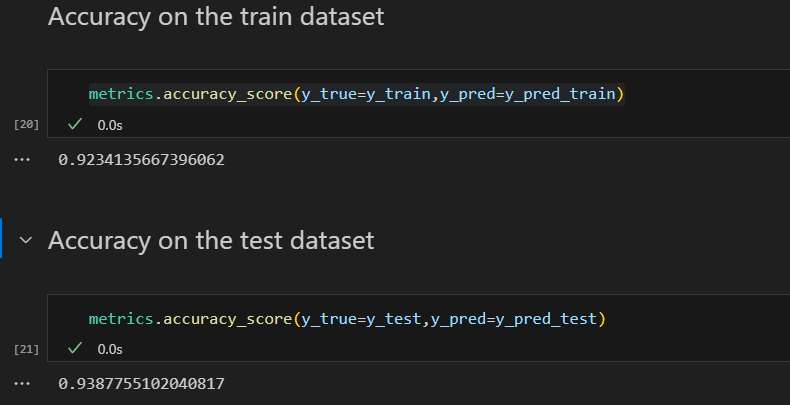
**Output:**

**Dataset 1:**

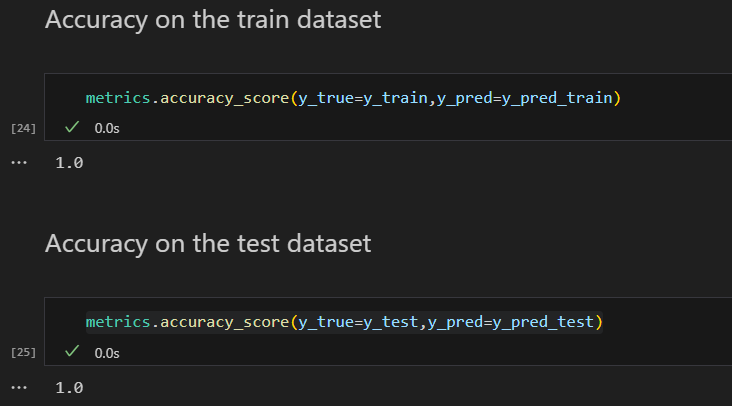
****

****

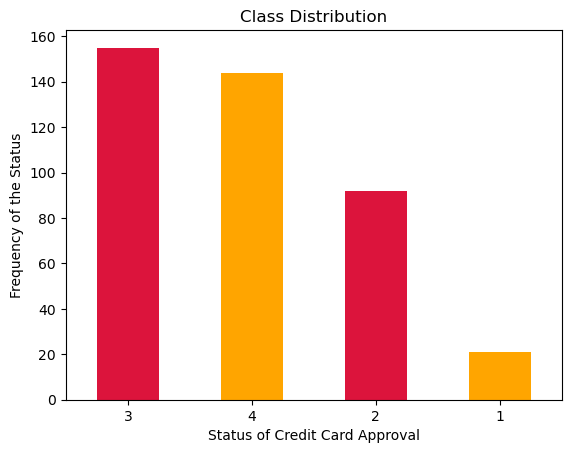
**Perceptron:**

****

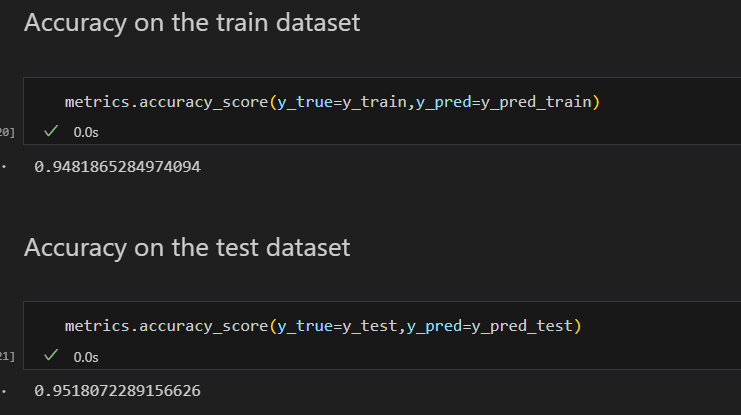
**Multilayer Perceptron:**

****

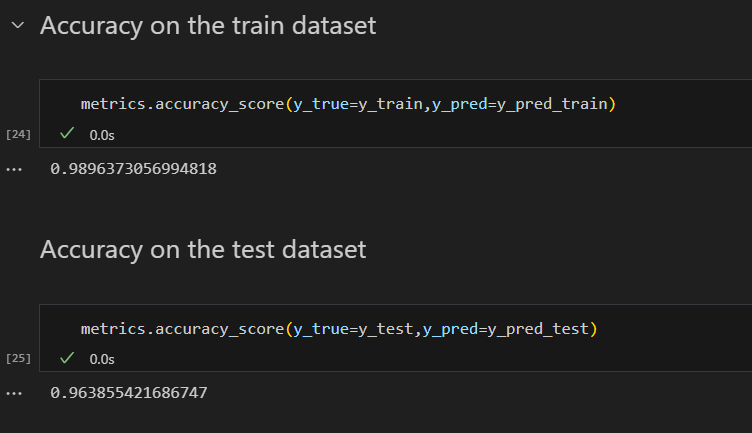
**Dataset 2:**

****

**Perceptron:**

****

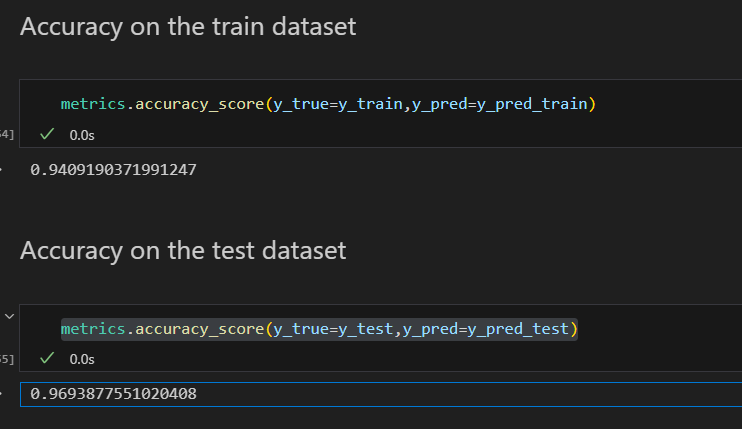
**Multilayer Perceptron:**

****

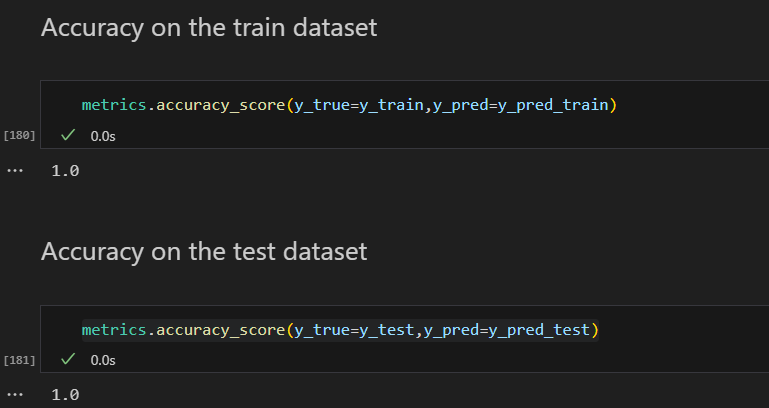
**B.3 Observations and learning:**

For Multilayer perceptron: **10 hidden layers**

clf = MLPClassifier(hidden\_layer\_sizes=(10), max\_iter=300,activation = 'relu',solver='adam',random\_state=1).fit(X\_train, y\_train)

****

**For 100 hidden layers:**

****

We observed increase in the accuracy after increasing the hidden layers. I even added few more parameters:

**hidden\_layer\_sizes*array-like of shape(n\_layers - 2,), default=(100,)***

The ith element represents the number of neurons in the ith hidden layer.

**activation*{‘identity’, ‘logistic’, ‘tanh’, ‘relu’}, default=’relu’***

Activation function for the hidden layer.

* ‘identity’, no-op activation, useful to implement linear bottleneck, returns f(x) = x
* ‘logistic’, the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
* ‘tanh’, the hyperbolic tan function, returns f(x) = tanh(x).
* ‘relu’, the rectified linear unit function, returns f(x) = max(0, x)

**solver*{‘lbfgs’, ‘sgd’, ‘adam’}, default=’adam’***

The solver for weight optimization.

* ‘lbfgs’ is an optimizer in the family of quasi-Newton methods.
* ‘sgd’ refers to stochastic gradient descent.
* ‘adam’ refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba

A graph of a graph of a graph

Description automatically generated with medium confidence

**B.4 Conclusion:**

Multilayer Perceptron gives us more parameters to play on the dataset and for better scalability also better accuracy as per increasing epochs/hidden layers . Successfully implemented perceptron and multilayer perceptron on the new dataset “cirrhosis.csv” classifying stages of cancer.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***