ASSIGNMENT-5

Name-Chanamolu Pranavya
ID- 700739974

NEURAL NETWORKS & DEEP LEARNING

GitHub link: https://github.com/pxc99740/Neural_networks_assignment_5.git

Videolink:
https://drive.google.com/drive/folders/1WQIZ6C5jhfxV0d9hkU73WhwZ4gtVDF7T

1.Implement Naïve Bayes method using scikit-learn library.
Use dataset available with name glass.
Use train_test_split to create training and testing part
Evaluate the model on test part using score and
classification report (y_true, y_pred).

I'm starting my program by importing the pandas and numpy libraries that are needed to generate arrays. Then, using functions like Pandas read_csv() method, which lets you work with files efficiently, I'm loading the data set by reading the glass.csv file.
Then, I'm using the df.head() method to display the data structure from the specified file. A specified number of rows, string from the top, are returned by the head() method.

X3=df.drop(["Type"], axis=1): This line drops the "Type" column from the dataframe 'df' and the new dataframe is assigned to the variable 'X3'.
y=df["Type"]: This line creates a new variable called "y" and assigns the "Type" column from the original dataframe called "df." Next, I created the training and testing portions using train_test_split.

Next, I have imported three modules from the scikit-learn library. Gaussian Naive Bayes: The Gaussian Naive Bayes technique for classification is implemented in this module. Metrics: For classification issues, this module offers a variety of Evaluation metrics, such as precision, recall, f1-score, and support.
Accuracy Score: This function calculates the accuracy of the classification model.

gnb = GaussianNB(): This line creates an instance of the Gaussian Naive Bayes algorithm.
y_pred = gnb.fit(X_train, y_train).predict(X_test): This line creates predictions for the test data (X_test) by using the predict method to train the model on the training set (X_train and y_train). The variable 'y_pred' contains the predictions.
acc_nb=accuracy_score(y_test,y_pred): This line calculates the accuracy of the model using the accuracy_score function, which was imported earlier. The accuracy is stored in the variable 'acc_nb'.

After computing the % and rounding it off to two digits, we get the Naïve Bayes accuracy as 53.49 and generated a classification report for the model as shown below.

```python
#Importing packages to create arrays
import pandas as pd
import numpy as np
```

[34] ✓ 0.0s Python

```python
#loading data set
df=pd.read_csv("glass.csv")
```

✓ 0.0s

```python
#to view the structure of data
df.head()
```

[36] ✓ 0.0s

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	1

```python
#removing the column to be predicted from the dataframe
X3=df.drop(["Type"],axis=1)

#creating an array for the target column prediction
y=df["Type"]
```

[37] ✓ 0.0s

```python
#splitting data(both X,y) for train & for test

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X3, y, test_size= 0.8 ,random_state=0)
```

[52] ✓ 0.0s

```
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
from sklearn.metrics import accuracy_score
```
[53] ✓ 0.0s

```
#1) Implement Naïve Bayes method using scikit-learn library
##Training data with Gaussian Naive Bayes
gnb = GaussianNB()
# Training on X_Train & y_Train
#Applying prediction on X_Test
y_pred = gnb.fit(X_train, y_train).predict(X_test)
# calculating the accuracy score(accuracy score is calculated on predicted value & Actual value)
acc_nb=accuracy_score(y_test,y_pred)
print("Naïve Bayes Accuracy is:", acc_nb)
```
[54] ✓ 0.0s

```
Naïve Bayes Accuracy is: 0.5348837209302325
```

```
#Calculating % and rounding off to 2 digits
print("Naïve Bayes Accuracy is:")
round(acc_nb * 100,2)
```
[68] ✓ 0.0s

```
Naïve Bayes Accuracy is:
53.49
```

```
## Classification report is used to measure the quality of predictions using classification algorithm and accuracy
print(metrics.classification_report(y_test,y_pred, zero_division=0))
```
[69] ✓ 0.0s

```
              precision    recall  f1-score   support

           1       0.58      0.23      0.33        60
           2       0.45      0.72      0.55        61
           3       0.00      0.00      0.00        15
           5       0.47      0.88      0.61         8
           6       0.55      1.00      0.71         6
           7       0.88      0.95      0.91        22

    accuracy                           0.53       172
   macro avg       0.49      0.63      0.52       172
weighted avg       0.52      0.53      0.48       172
```

2.Implement linear SVM method using scikit library Use the same dataset above Use train_test_split to create training and testing part.

In this program, I have utilized the equivalent dataset utilized above and the means I have taken to import libraries and read the dataset are referenced above being referred to question 1. I have then imported two modules from the scikit-learn library. SVC: The Support Vector Classification algorithm is put into action in this module. LinearSVC: This module carries out the Linear Support Vector Classification algorithm, which is a variety of the SVC calculation that utilizes a linear decision boundary rather than a non-linear boundary.

svc.fit(X_train, y_train): The model is trained using the training data (X_train and y_train) on this line.

y_pred = svc.predict(X_test): This line uses the predict method to generate predictions for the test data (X_test). The predictions are stored in the variable 'y_pred'. I have then calculated the accuracy of the model using the accuracy_score function. Upon calculating % and rounding off to 2 digits the SVM accuracy we get is 62.61 and the generated classification report is as shown below:

```
#2) Implement linear SVM method using scikit library
# importing packages for SVM.SVC
from sklearn.svm import SVC, LinearSVC

#Training data with Support vector Classification
svc = SVC(kernel ='linear', C = 5)
# Training on X_Train & y_Train
svc.fit(X_train, y_train)
#Applying prediction on X_Test
y_pred = svc.predict(X_test)
# calculating the accuracy score(accuracy score is calculated on predicted value & Actual value)
acc_svc=accuracy_score(y_test,y_pred)
print("LinearSVM Accuracy is:", acc_svc)
```

[70] ✓ 0.0s

```
LinearSVM Accuracy is: 0.622093023255814
```

```
#Calculating % and rounding off to 2 digits
print("SVM Accuracy is:")
round(acc_svc * 100 , 2)
```

[71] ✓ 0.0s

```
SVM Accuracy is:

62.21
```

```
## Classification report is used to measure the quality of predictions using classification algorithm and accuracy
print(metrics.classification_report(y_test,y_pred, zero_division=0))
```

[85] ✓ 0.0s

```
              precision    recall  f1-score   support

           1       0.67      0.62      0.64        60
           2       0.54      0.67      0.60        61
           3       0.00      0.00      0.00        15
           5       0.47      0.88      0.61         8
           6       0.67      0.67      0.67         6
           7       0.95      0.82      0.88        22

    accuracy                           0.62       172
   macro avg       0.55      0.61      0.57       172
weighted avg       0.59      0.62      0.60       172
```

```
# For classification problem, the best score is 100% accuracy.
# In Gaussian Naive Bayes accuracy score is 53.49%
# In Support vector Classification accuracy score is 62.61%

# So in this data set Linear SVM is better than Gaussian Naive Bayes,
# as accuracy score of Linear Svm is highest.
```

[86] ✓ 0.0s

Which algorithm you got better accuracy? Can you justify why?

Linear SVM has a better accuracy score than Gaussian Naive Bayes with 62.61% compared to 53.49%. One possible explanation for this difference in performance could be attributed to the suitability of Linear Support Vector Machines (SVM) for handling complex and non-linearly separable datasets. SVM aims to maximize the margin between data points and the decision boundary, which enhances its resilience to outliers and non-linear relationships within the data. Moreover, SVM offers flexibility through various kernel functions, enabling it to adapt to different data distributions effectively. Conversely, Gaussian Naive Bayes assumes feature independence, which may limit its effectiveness, particularly in handling complex datasets.

Additionally, the effectiveness of Naive Bayes can be influenced by factors such as dataset size, structure, and the distribution of data. For instance, if the features are not independent as assumed by Naive Bayes, its performance may suffer. Furthermore, suboptimal hyperparameters in Naive Bayes can lead to lower accuracy scores, highlighting the importance of proper parameter tuning. It's crucial to consider various evaluation metrics beyond just accuracy, such as precision, recall, and F1-score, for a comprehensive assessment of model performance.