Neural Networks and Deep Learning – ICP 5

GitHub Link: https://github.com/srija1609/NNDL_ICP-5
Name: Baby Srija Bitra
Student ID: 700755908

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).

In this program, I am first Importing the required packages to create arrays which are pandas and numpy. Then I am loading the data set by reading the giving glass.csv file by using the Functions like the Pandas read_csv() method which enable you to work with files effectively. Then I am displaying the structure of data from given file, using the df.head() method. The head () method returns a specified number of rows, string from the top.

X2=df.drop(["Type"], axis=1): This line drops the "Type" column from the dataframe 'df' and assigns the new dataframe to the variable 'X2'. The 'axis' parameter is set to 1, indicating that a column is being dropped, rather than a row.

y=df["Type"]: This line extracts the "Type" column from the original dataframe 'df' and assigns it to a new variable 'y'. Then I have used train_test_split to create training and testing part.

I have then imported three modules from the scikit-learn library. GaussianNB: This module implements the Gaussian Naive Bayes algorithm for classification. metrics: This module provides a range of evaluation metrics for classification problems, including 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 trains the model on the training data (X_train and y_train) and uses the predict method to generate predictions for the test data (X_test). The predictions are stored in the variable 'y_pred'.

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'.

Upon Calculating 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.

In [339]:
```
#Importing packages to create arrays
import pandas as pd
import numpy as np
```

In [340]:
```
#loading data set
df=pd.read_csv("glass.csv")
```

In [341]:
```
#to view the structure of data
df.head()
```

Out[341]:

	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

In [342]:
```
#removing the column to be predicted from the dataframe
X2=df.drop(["Type"],axis=1)

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

In [616]:
```
#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(X2, y, test_size= 0.8 ,random_state=0)
```

In [617]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
from sklearn.metrics import accuracy_score
```

In [618]:
```
#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)
```

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

In [619]:
```
#Calculating % and rounding off to 2 digits
print("Naïve Bayes Accuracy is:")
round(acc_nb * 100,2)
```

```
Naïve Bayes Accuracy is:
```

Out[619]:
```
53.49
```

In [620]:
```
## 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))
```

```
              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 used the same dataset used above and the steps I have taken to import libraries and read the dataset are mentioned above in question 1. I have then imported two modules from the scikit-learn library. SVC: This module implements the Support Vector Classification algorithm. LinearSVC: This module implements the Linear Support Vector Classification algorithm, which is a variation of the SVC algorithm that uses a linear decision boundary instead of a non-linear boundary. svc.fit(X_train, y_train): This line trains the model on the training data (X_train and y_train). 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:

In [621]:
```
#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)
```

```
LinearSVM Accuracy is: 0.622093023255814
```

In [622]:
```
#Calculating % and rounding off to 2 digits
print("SVM Accuracy is:")
round(acc_svc * 100 , 2)
```

```
SVM Accuracy is:
```

Out[622]:
```
62.21
```

In [623]:
```
## 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))
```

```
              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
```

In [586]:
```
# 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.
```

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%. The reason for this could be because Linear SVM is better suited for complex and non-linearly separable datasets. The SVM algorithm tries to find the maximum margin between the data points and the decision boundary, which makes it robust to outliers and non-linearities. Additionally, SVM allows for more flexibility in terms of choosing different kernel functions to tackle different types of data distributions. Gaussian Naive Bayes, on the other hand, assumes independence between the features and is not as robust as SVM for complex datasets.

The dataset size and structure, if the data has many features and is easy to interpret, Naive Bayes can be more effective.

The data distribution, Naive Bayes assumes that the features are independent, if the features in the data are not independent, the performance of Naive Bayes can decrease.

The accuracy of the model can also depend on the choice of hyperparameters, if the hyperparameters of the Naïve Bayes algorithm were not optimized, this could result in a lower accuracy score.
It is important to keep in mind that accuracy is not the only evaluation metric that should be considered, other metrics such as precision, recall, and F1-score should also be considered to get a comprehensive evaluation of the model's performance.