Fall 2023: CS5720

Neural Networks & Deep Learning - ICP-5

Name: Manisha Lakkarsu

Student Id: 700746573

Git link: https://github.com/Mani543/Manisha_NNDL_ICP5.git

Video link: https://drive.google.com/file/d/1-b9UwmetdWR82zKFljlGe8CURC8LqPcm/view?usp=sharing

1. Implement Naive 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).

- Using dataset available with name glass.

In [3]:
```
# Use dataset available with name glass.
data_frame = pd.read_csv("glass.csv")
data_frame.head()
```

Out[3]:

	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

- Use train_test_split to create training and testing part.

In [5]:
```
# seperating a_data and b_data
b_data = data_frame['Type']
a_data = data_frame.drop('Type', axis=1)
a_data.head()
```

Out[5]:

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

In [14]:
```
# Use train_test_split to create training and testing part
a_train, a_test, b_train, b_test = train_test_split(a_data, b_data, test_size=0.3, random_state=7)
```

- Shaping test and train data and predicting a_test data using Naïve Bayes method.

In [15]:
```
# Shape train data
print(a_train.shape, b_train.shape)
```

```
(149, 9) (149,)
```

In [16]:
```
# Shape test data
print(a_test.shape, b_test.shape)
```

```
(65, 9) (65,)
```

In [17]:
```
# Training Naive Bayes Model
naive_bayes_model = GaussianNB()
naive_bayes_model.fit(a_train, b_train)
```

Out[17]:

GaussianNB

GaussianNB()

In [18]:
```
# Predicting the a_test data using Naive Bayes Model
b_prediction = naive_bayes_model.predict(a_test)
print(b_prediction)
```

- Evaluate the model on test part using score.

In [19]:
```
# Naive Bayes Model score
print(naive_bayes_model.score(a_test, b_test))
```

```
0.24615384615384617
```

- Classify the report.

```
In [21]: # Classification report of Naive Bayes Model
         print(classification_report(b_test, b_prediction))
```

```
              precision    recall  f1-score   support

           1       0.33      0.10      0.15        20
           2       0.60      0.21      0.31        29
           3       0.03      0.25      0.05         4
           5       0.00      0.00      0.00         4
           6       0.00      0.00      0.00         1
           7       0.88      1.00      0.93         7

    accuracy                           0.25        65
   macro avg       0.31      0.26      0.24        65
weighted avg       0.47      0.25      0.29        65
```

2. Implement linear SVM method using scikit library. Use the same dataset above. 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).

- Training SVM model using above data and predict the results.

```
In [22]: # Linear SVM Model training
         svm_model = LinearSVC(random_state=6)
         svm_model.fit(a_train, b_train)
```

```
Out[22]: LinearSVC
         LinearSVC(random_state=6)
```

```
In [25]: # Predicting the a_test data using Linear SVM Model
         b_prediction = svm_model.predict(a_test)
         print(b_prediction)
```

```
[2 1 2 2 1 1 2 2 2 1 1 1 1 2 1 1 1 6 2 6 1 2 2 7 2 7 7 1 2 2 7 2 1 2 2 7 1
 2 2 2 2 7 5 2 2 7 1 2 2 2 1 2 2 1 2 6 2 2 6 2 2 2 1 7 2]
```

- Evaluate the model on test part using score.

```
In [30]: # Linear SVM Model score
         print(svm_model.score(a_test, b_test))
```

```
0.5384615384615384
```

- Classify the report.

```
In [33]: # Classification report of Linear SVM Model
         print(classification_report(b_test, b_prediction))
```

```
              precision    recall  f1-score   support

           1       0.50      0.45      0.47        20
           2       0.56      0.66      0.60        29
           3       0.00      0.00      0.00         4
           5       0.00      0.00      0.00         4
           6       0.00      0.00      0.00         1
           7       0.88      1.00      0.93         7

    accuracy                           0.54        65
   macro avg       0.32      0.35      0.34        65
weighted avg       0.50      0.54      0.52        65
```

- Which algorithm you got better accuracy? Can you justify why?
    - Linear SVM has better accuracy than Naive Bayes Model because SVM can perform well in classifying multi-dimensional data and since Naive Bayes is based upon the frequency of occurrence it was not able to classify data.