ADVANCED MACHINE LEARNING

ASSIGNMENT – 1

SUBMITTED BY

AJITH RAJ PERIYASAMY

Github Link:

https://github.com/Ajith0719/aperiyas_64061/tree/main/Assignment%201

Task 1:

The first part of the assignment required us to Read, run, and understand the KNN python workings that Professor assigned.

A case study using iris dataset for KNN algorithm

```
[ ] # import modules for this project
    from sklearn import datasets
    from sklearn.metrics import accuracy_score
    from sklearn.model_selection import train_test_split

    # load iris dataset
    iris = datasets.load_iris()
    data, labels = iris.data, iris.target

    # training testing split
    res = train_test_split(data, labels,
                           train_size=0.8,
                           test_size=0.2,
                           random_state=12)
    train_data, test_data, train_labels, test_labels = res
```

The **First** step in the given template is to import the required python modules for the KNN analysis such as dataset, accuracy score which is the performance metrics, and train_test_split which will help us partition our dataset. A variable named 'Iris' is created and the Iris dataset has been assigned to it and the KNN analysis will be performed on this dataset.

The **Second** step is to split the Iris dataset into Train [80%] and Test [20%]. The Train set will be used to train the model, and the trained model would be tested on the test data.

```
# Create and fit a nearest-neighbor classifier
from sklearn.neighbors import KNeighborsClassifier
# classifier "out of the box", no parameters
knn = KNeighborsClassifier()
knn.fit(train_data, train_labels)

# print some interested metrics
print("Predictions from the classifier:")
learn_data_predicted = knn.predict(train_data)
print(learn_data_predicted)
print("Target values:")
print(train_labels)
print(accuracy_score(learn_data_predicted, train_labels))

# re-do KNN using some specific parameters.
knn2 = KNeighborsClassifier(algorithm='auto',
                            leaf_size=30,
                            metric='minkowski',
                            p=2,         # p=2 is equivalent to euclidian distance
                            metric_params=None,
                            n_jobs=1,
                            n_neighbors=5,
                            weights='uniform')

knn.fit(train_data, train_labels)
test_data_predicted = knn.predict(test_data)
accuracy_score(test_data_predicted, test_labels)
```

The **Third** step is to perform KNN analysis on the Iris dataset. Initially a KNN classifier with no parameters is constructed. Then specific parameters are defined and the same is repeated. Finally, accuracy values are computed and printed out.

Task 2:

The second part of the assignment is to conduct a similar KNN analysis using a simulated dataset.

```
# import the required python modules
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split

# Generate a simulated dataset
centers = [[2, 4], [6, 6], [1, 9]]
n_classes = len(centers)
data, labels = make_blobs(n_samples=150,
                          centers=np.array(centers),
                          random_state=1)
```

The **First** step is to import the required python modules for the KNN analysis such as generating simulated dataset, matplotlib to create visualizations, numpy to work with arrays, and train_test_split for data partitioning. The **Second** step is to generate a simulated dataset for our KNN analysis.

```
# do a 80-20 split of the data
res = train_test_split(data, labels,
                       train_size=0.8,
                       test_size=0.2,
                       random_state=1)
train_data, test_data, train_labels, test_labels = res
```

The **Third** step is to partition the generated dataset. We use an 80-20 split method of partitioning. The data is split into Train [80%] which will be used to train the model, and Test [20%] which will be used to test the model and random seed is a component that ensures code reproducibility.

```
# perform a KNN analysis of the simulated data
from sklearn.neighbors import KNeighborsClassifier
# classifier without any parameter
knn = KNeighborsClassifier()
knn.fit(train_data,train_labels)
```

The next step is to perform the KNN analysis on our simulated dataset without specifying the parameters.

```
# printing few metrics
from sklearn.metrics import accuracy_score
print("Predictions from the classifier:")
learnt_data_predicted = knn.predict(train_data)
print(learnt_data_predicted)
print("target values:")
print(train_labels)
print(accuracy_score(learnt_data_predicted,train_labels))
```

As a next step we print out few metrics such as predictions on train data, and their accuracy scores.

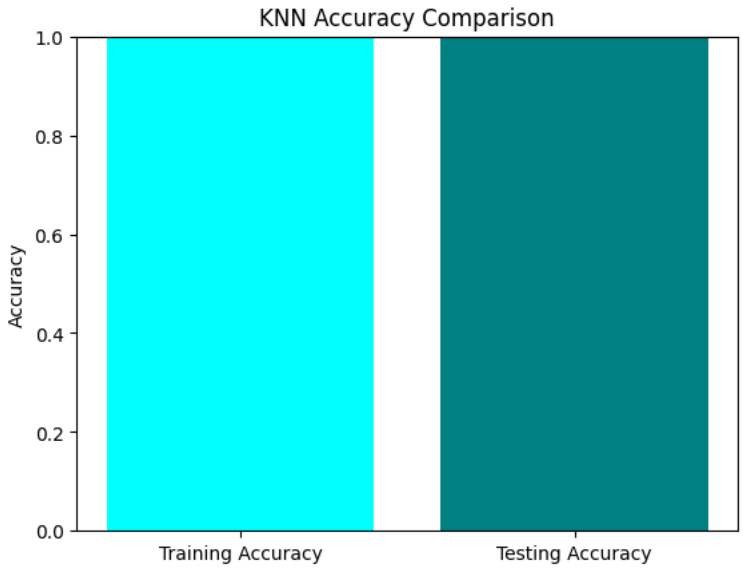
```
# performing KNN analysis with specific customized parameters
knn2 = KNeighborsClassifier(n_neighbors=5, weights = 'unifrom',algorithm = 'auto',leaf_size=30,p=2,metric='minkowski',metric_params=None,n_jobs=None)
knn.fit(train_data, train_labels)

# Accuracy for training set
train_predictions = knn.predict(train_data)
train_accuracy = accuracy_score(train_predictions, train_labels)

# Accuracy for testing set
test_predictions = knn.predict(test_data)
test_accuracy = accuracy_score(test_predictions, test_labels)
```

We now perform KNN with specified parameters such as n_neighbors, weights, algorithm, leaf_size, p, metric, metric_params, and n_jobs. Note that setting the P value as 2 denotes that the Euclidean distance method is applied here for computing distance. After performing KNN analysis, now compute the accuracy scores for both the training and the testing set.

The Output:

It is evident from the analysis that the accuracy on both the train and test set is 1 or 100%. This suggests that the model has perfectly learnt the training set and is doing an excellent job in predicting the values of the unknown test set. Given the set parameters and also considering the well-distributed nature of the simulated dataset, we were able to achieve the accuracy of 100%.

Visualizing the output:



We used Matplotlib from python to plot out accuracy results. The above bar graph denotes that the accuracy on both the train and test set is 1 or 100%.