Assignment 1
Advanced Machine learning

Instructor: Chaojiang (CJ) Wu, Ph.D.
Department of MIS
Kent state university.

Student: Lava Kumar Saripudi
Reg. ID: 811196583

Contents:

Problem Statement ---------- 02

Data Analysis---------- 03

Conclusion ----------08

Problem statement:

Use the following code to generate an artificial dataset which contain three classes. Conduct a similar KNN analysis to the dataset and report your accuracy.

```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np

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)
```

In this assignment I have done the analysis as instructed by following the below steps

1) do an 80-20 split of the data

2) perform a KNN analysis of the simulated data

3) output accuracy score

4) plot your different results

Dataset Information:

Here I used the make_blobs() function which is available in sklearn.datasets and created an artificial data set with 600 samples and created 3 classes for the purpose of analysis

```
centers = [[2, 4], [6, 6], [1, 9]]
n_classes = len(centers)
data, labels = make_blobs(n_samples=600,
                          centers=np.array(centers),
                          random_state=1)
```

The make_blobs() function can be used to generate blobs of points with a Gaussian distribution.

You can control how many blobs to generate and the number of samples to generate, as well as a host of other properties.

The problem is suitable for linear classification problems given the linearly separable nature of the blobs.

Data Analysis:

The first step will be loading or importing the required modules for the data analysis as shown below.

```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

1)do an 80-20 split of the data

Here to train and test the data we are splitting it in to 80% train data and 20% test data

```
# do a 80-20 split of the data
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
```

2) perform a KNN analysis of the simulated data

Here to perform KNN analysis we are using KNeighborsClassifier().

K Neighbors Classifier

- K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most like the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much like the new data.

KNN analysis without parameters:

```
#perform a KNN analysis of the simulated data

# Create and fit a nearest-neighbor classifier
from sklearn.neighbors import KNeighborsClassifier

# classifier "out of the box", no parameters'
for k in neighbors:
  knn = KNeighborsClassifier()
  print("K-Value :", k)
  knn.fit(train_data, train_labels)
  print("Predictions from the classifier:")
  learn_data_predicted = knn.predict(train_data)
  print(learn_data_predicted)
  print("Target values:")
  print(train_labels)
 # output accuracy score
  print(accuracy_score(learn_data_predicted, train_labels))
  #Validation
  print("test-KNN")
  score = knn.score(test_data, test_labels)
  scores.append(score)
  print(score)
```

Here I have taken K values as k = 1,2,3,4 and validated the trained data result with test data result.

Output:

```
K-Value : 1
Predictions from the classifier:
[0 1 2 2 1 2 0 2 0 0 0 0 2 2 1 1 0 0 1 2 1 1 0 2 2 2 1 1 2 1 0 0 2 2 2 0 0
 1 1 0 2 2 1 1 1 0 1 1 0 2 1 0 0 0 2 1 0 2 2 2 0 2 2 1 1 0 1 1 2 0 1 1 0 1
 2 0 2 1 1 0 2 2 0 1 2 0 0 2 2 1 1 2 2 1 1 0 1 0 2 1 0 2 1 0 0 1 1 2 1 0 2
 1 0 0 1 1 0 1 1 2 1 1 1 2 0 0 2 1 0 0 2 2 2 0 0 1 1 2 2 0 0 1 1 0 2 2 1 1
 2 0 0 2 0 2 2 0 1 0 2 1 2 0 0 1 1 0 2 2 1 2 0 0 1 0 1 2 2 2 0 0 0 1 0 1 2
 2 2 0 1 1 1 1 0 1 0 2 2 2 1 0 1 1 2 0 0 1 1 0 2 2 1 1 1 1 0 0 0 0 1 1 2 2
 2 0 2 0 2 0 2 1 2 1 0 1 1 2 2 0 1 1 1 1 2 0 2 2 1 0 2 0 1 2 1 0 2 0 1 2 1
 2 2 2 0 1 2 0 0 1 0 1 0 2 2 0 2 2 0 1 0 2 2 0 1 1 0 2 1 0 0 0 0 2 0 2 0 0
 0 0 2 1 1 0 2 2 0 0 1 2 2 2 2 2 2 2 2 2 2 1 0 1 1 0 1 2 0 0 0 1 1 1 2 0 1
 2 0 1 2 2 0 1 2 0 2 2 2 2 1 2 2 2 0 1 2 0 1 1 0 1 1 0 0 1 1 1 0 1 1 1 1 1
 1 2 0 0 2 0 1 2 0 2 2 0 2 2 2 2 2 2 2 1 2 2 0 1 1 1 2 1 1 1 1 0 0 2 0 0 0 1
 2 2 2 1 2 2 0 0 1 1 0 2 1 2 1 1 2 1 2 2 0 1 0 1 1 0 1 0 2 0 0 2 2 0 1 0 1
 0 2 0 1 0 1 1 1 2 2 2 0 2 1 1 1 1 2 2 0 2 2 0 1 1 1 0 2 1 1 1 0 2 0 0 0]
Target values:
[0 1 2 2 1 2 0 2 0 0 0 0 2 2 1 1 0 0 1 2 1 1 0 2 2 2 1 1 2 1 0 0 2 2 2 0 0
 1 1 0 2 2 1 1 1 0 1 1 0 2 1 0 0 0 2 1 0 2 2 2 0 2 2 1 1 0 1 1 2 0 1 1 0 1
 2 0 2 1 1 0 2 2 0 1 2 0 0 2 2 1 1 2 2 1 1 0 1 0 2 1 0 2 1 0 0 1 1 2 1 0 2
 1 0 0 1 1 0 1 1 2 1 1 1 2 0 0 2 1 0 0 2 2 2 0 0 1 1 2 2 0 0 1 1 0 2 2 1 1
 2 0 0 2 0 2 2 0 1 0 2 1 2 0 0 1 1 0 2 2 1 2 0 0 1 0 1 2 2 2 0 0 0 1 0 1 2
 2 2 0 1 0 1 1 0 1 0 2 2 2 1 0 1 1 2 0 0 1 1 0 2 2 1 1 1 1 0 0 0 0 1 1 2 2
 2 0 2 0 2 0 2 1 2 1 0 1 1 2 2 1 1 1 1 1 2 0 2 2 1 0 2 0 1 2 1 0 2 0 1 2 1
 2 2 2 0 1 2 0 0 1 0 1 0 2 2 1 2 2 0 1 0 2 2 0 1 1 0 2 1 1 0 0 0 2 0 2 0 0
 0 0 2 1 1 0 2 2 0 0 1 2 2 2 2 2 2 2 2 2 2 1 0 1 1 0 1 2 0 0 0 1 1 1 2 0 1
 2 0 1 2 2 0 1 2 0 2 2 2 2 1 2 2 2 0 1 2 0 1 1 0 1 1 0 0 1 1 1 0 1 1 1 1 1
 1 2 0 0 2 0 1 2 0 2 2 0 2 2 2 2 2 2 2 1 2 2 0 1 1 1 2 1 1 1 1 0 0 2 0 0 0 1
 2 2 2 1 2 2 0 0 1 1 0 2 1 2 1 1 2 1 2 2 0 1 0 1 1 0 1 0 2 0 0 2 2 0 1 0 1
 0 2 0 1 0 1 1 1 2 2 2 0 2 1 1 1 1 2 2 0 2 2 0 1 1 1 0 2 1 1 1 0 2 0 0 0]
0.9895833333333334
test-KNN
0.975
```

KNN analysis with parameters:

```
# re-do KNN using some specific parameters.
for k in neighbors:
  knn = KNeighborsClassifier(algorithm='auto',
                             leaf_size=40,
                             metric='minkowski',
                             p=2,         # p=2 is equivalent to euclidian distance
                             metric_params=None,
                             n_jobs=1,
                             n_neighbors=k,
                             weights='uniform')
  print("K-Value :", k)
  knn.fit(train_data, train_labels)
  print("Predictions from the classifier:")
  learn_data_predicted = knn.predict(train_data)
  print(learn_data_predicted)
  print("Target values:")
  print(train_labels)
   # output accuracy score
  print(accuracy_score(learn_data_predicted, train_labels))
  #Validation
  print("test-KNN")
  score = knn.score(test_data, test_labels)
  scores.append(score)
  print(score)
```

Output:

```
K-Value : 1
Predictions from the classifier:
[0 1 2 2 1 2 0 2 0 0 0 0 2 2 1 1 0 0 1 2 1 1 0 2 2 2 1 1 2 1 0 0 2 2 2 0 0
 1 1 0 2 2 1 1 1 0 1 1 0 2 1 0 0 0 2 1 0 2 2 2 0 2 2 1 1 0 1 1 2 0 1 1 0 1
 2 0 2 1 1 0 2 2 0 1 2 0 0 2 2 1 1 2 2 1 1 0 1 0 2 1 0 2 1 0 0 1 1 2 1 0 2
 1 0 0 1 1 0 1 1 2 0 1 1 2 0 0 2 1 0 0 2 2 2 0 0 1 1 2 2 0 0 1 1 0 2 2 1 1
 2 0 0 2 0 2 2 0 1 0 2 1 2 0 0 1 1 0 2 2 1 2 0 0 1 0 1 2 2 2 0 0 0 1 0 1 2
 2 2 0 1 0 1 1 0 1 0 2 2 2 1 0 1 1 2 0 0 1 1 0 2 2 1 1 1 1 0 0 0 0 1 1 2 2
 2 0 2 0 2 0 2 1 2 1 0 1 1 2 2 1 1 1 1 1 2 0 2 2 1 0 2 0 1 2 1 0 2 0 1 2 1
 2 2 2 0 1 2 0 0 1 0 1 0 2 2 1 2 2 0 1 0 2 2 0 1 1 0 2 1 1 0 0 0 2 0 2 0 0
 0 0 2 1 1 0 2 2 0 0 1 2 2 2 2 2 2 2 2 2 1 0 1 1 0 1 2 0 0 0 1 1 1 2 0 1
 2 0 1 2 2 0 1 2 0 2 2 2 2 1 2 2 2 0 1 2 0 1 1 0 1 1 0 0 1 1 1 0 1 1 1 1 1
 1 2 0 0 2 0 1 2 0 2 2 0 2 2 2 2 2 2 1 2 2 0 1 1 1 2 1 1 1 1 0 0 2 0 0 0 1
 2 2 2 1 2 2 0 0 1 1 0 2 1 2 1 1 2 1 2 2 0 1 0 1 1 0 1 0 2 0 0 2 2 0 1 0 1
 0 2 0 1 0 1 1 1 2 2 2 0 2 1 1 1 1 2 2 0 2 2 0 1 1 1 0 2 1 1 1 0 2 0 0 0]
Target values:
[0 1 2 2 1 2 0 2 0 0 0 0 2 2 1 1 0 0 1 2 1 1 0 2 2 2 1 1 2 1 0 0 2 2 2 0 0
 1 1 0 2 2 1 1 1 0 1 1 0 2 1 0 0 0 2 1 0 2 2 2 0 2 2 1 1 0 1 1 2 0 1 1 0 1
 2 0 2 1 1 0 2 2 0 1 2 0 0 2 2 1 1 2 2 1 1 0 1 0 2 1 0 2 1 0 0 1 1 2 1 0 2
 1 0 0 1 1 0 1 1 2 0 1 1 2 0 0 2 1 0 0 2 2 2 0 0 1 1 2 2 0 0 1 1 0 2 2 1 1
 2 0 0 2 0 2 2 0 1 0 2 1 2 0 0 1 1 0 2 2 1 2 0 0 1 0 1 2 2 2 0 0 0 1 0 1 2
 2 2 0 1 0 1 1 0 1 0 2 2 2 1 0 1 1 2 0 0 1 1 0 2 2 1 1 1 1 0 0 0 0 1 1 2 2
 2 0 2 0 2 0 2 1 2 1 0 1 1 2 2 1 1 1 1 1 2 0 2 2 1 0 2 0 1 2 1 0 2 0 1 2 1
 2 2 2 0 1 2 0 0 1 0 1 0 2 2 1 2 2 0 1 0 2 2 0 1 1 0 2 1 1 0 0 0 2 0 2 0 0
 0 0 2 1 1 0 2 2 0 0 1 2 2 2 2 2 2 2 2 2 1 0 1 1 0 1 2 0 0 0 1 1 1 2 0 1
 2 0 1 2 2 0 1 2 0 2 2 2 2 1 2 2 2 0 1 2 0 1 1 0 1 1 0 0 1 1 1 0 1 1 1 1 1
 1 2 0 0 2 0 1 2 0 2 2 0 2 2 2 2 2 2 1 2 2 0 1 1 1 2 1 1 1 1 0 0 2 0 0 0 1
 2 2 2 1 2 2 0 0 1 1 0 2 1 2 1 1 2 1 2 2 0 1 0 1 1 0 1 0 2 0 0 2 2 0 1 0 1
 0 2 0 1 0 1 1 1 2 2 2 0 2 1 1 1 1 2 2 0 2 2 0 1 1 1 0 2 1 1 1 0 2 0 0 0]
1.0
test-KNN
0.9666666666666667
```

3) Output accuracy score:

After the KNN analysis, we get the accuracy of 0.9895833333333334 for the trained data and get an accuracy of 0.975 on the test data without using parameters.

```
 0 2 0 1 0 1 1 1 2 2 2 0 2 1 1 1
0.9895833333333334
test-KNN
0.975
```

We get an accuracy of 0.98 for the trained data and get an accuracy of 0.98 with the test data with using parameters.

```
 0 2 0 1 0 1 1 1 2 1
0.9854166666666667
test-KNN
0.9833333333333333
```

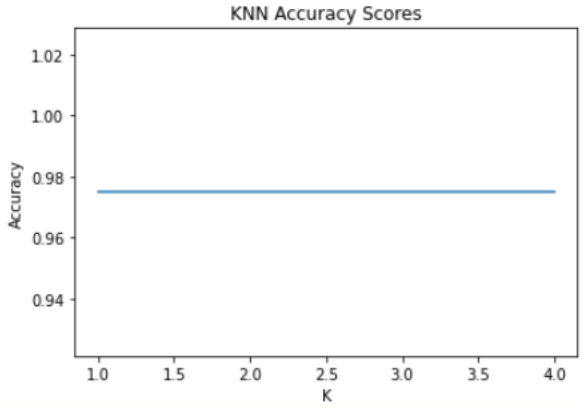
4) plot your different results:

Here we have plotted the graphs with K values and test results.

Without Parameters:

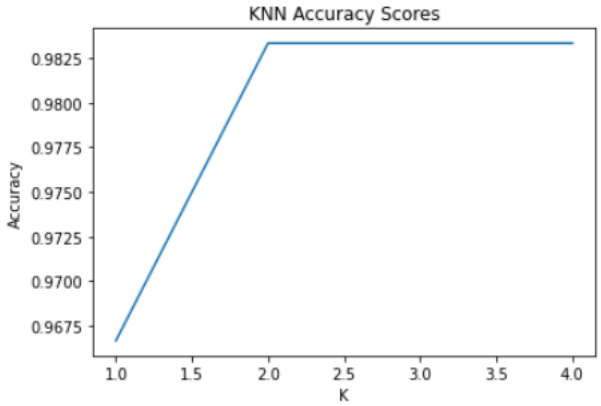
```
# plot your different results

# Plot the accuracy scores for each K value
plt.plot(neighbors, scores)
plt.title("KNN Accuracy Scores")
plt.xlabel("K")
plt.ylabel("Accuracy")
plt.show()
```



With Parameters:

```
# plot your different results
# Plot the accuracy scores for each K value
plt.plot(neighbors, scores)
plt.title("KNN Accuracy Scores")
plt.xlabel("K")
plt.ylabel("Accuracy")
plt.show()
```



Conclusion:

We can conclude by saying that we have performed a KNN analysis by training and testing the artificially produced dataset with a split of 80 - 20 and acquired the accuracy of about 98% and plotted the respective plots for the same.