JawadAhmed_20P-0165

February 11, 2023

1 Task 1.

Develop the K-Nearest Neighbors (KNN) algorithm from scratch using Python, without relying on any libraries.

[1]:
```
import numpy as np
from collections import Counter

class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

#     Function will take two set of values as parameter and calculate eculidean␣
↪distance
    def euclidean_distance(self, x1, x2):
        return np.sqrt(np.sum((x1 - x2)**2))

#     Function will take two set of values as parameter and calculate manhattan␣
↪distance
    def manhattan_distance(self, a, b):
        return sum(abs(val1-val2) for val1, val2 in zip(a,b))

#         Return the Count of each value present in list
    def counter(self, lst):
        count = {}
        for item in lst:
            if item in count:
                count[item] += 1
            else:
                count[item] = 1
        return count

#     return the most_common value (or the result by processing different␣
↪neigbors)
```

```
    def most_common(self, d, n=None):
        items = [(v, k) for k, v in d.items()]
        items.sort(reverse=True)
        if n is None:
            n = len(items)
        return [(k, v) for v, k in items[:n]]



    def predict(self, X, distance_type='euc'):
        y_pred = []
        for x in X:
            if (distance_type == 'euc'):
                distances = [self.euclidean_distance(x, x_train) for x_train in␣
 ↪self.X_train]
            elif (distance_type == 'man'):
                distances = [self.manhattan_distance(x, x_train) for x_train in␣
 ↪self.X_train]
            # argsort (return sorted indices)
            # k_neighbors stores neighbors in ascending order
            k_indices = np.argsort(distances)[:self.k]
            k_neighbors = [self.y_train[i] for i in k_indices]
            counts = self.counter(k_neighbors)
            most_com = self.most_common(counts, 1)
            y_pred.append(most_com[0][0])
        return y_pred

# test cases
knn = KNN(k=3)
X_train = np.array([[7,7], [7,4], [3, 4], [1, 4]])
y_train = np.array([0,0,1,1])
knn.fit(X_train, y_train)

X_test = np.array([[3, 7], [1, 1]])
y_test = np.array([1, 1])
y_pred = knn.predict(X_test, 'man')

print("Accuracy: ", np.mean(y_pred == y_test))
```

```
Accuracy:  1.0
```

2 Task 2. Utilizing the fruit_data_with_colors.csv dataset, perform the following steps:

3 a. Read and load the data into the program.

[2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

df = pd.read_csv("fruit_data_with_colors _1_.csv")
df.head()
```

[2]:

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192.0	8.4	7.3	0.55
1	1	apple	granny_smith	180.0	8.0	6.8	0.59
2	1	apple	granny_smith	176.0	7.4	7.2	0.60
3	2	mandarin	mandarin	86.0	6.2	4.7	0.80
4	2	mandarin	mandarin	84.0	6.0	4.6	0.79

4 b. Prepare the data by eliminating any features that contain text or categorical values

[3]:
```
df.drop(['fruit_label', 'fruit_subtype'], axis=1, inplace=True)
df
```

[3]:

	fruit_name	mass	width	height	color_score
0	apple	192.0	8.4	7.3	0.55
1	apple	180.0	8.0	6.8	0.59
2	apple	176.0	7.4	7.2	0.60
3	mandarin	86.0	6.2	4.7	0.80
4	mandarin	84.0	6.0	4.6	0.79
5	mandarin	80.0	5.8	4.3	0.77
6	mandarin	80.0	5.9	4.3	0.81
7	mandarin	76.0	5.8	4.0	0.81
8	apple	178.0	7.1	7.8	0.92
9	apple	172.0	7.4	7.0	0.89
10	apple	166.0	6.9	7.3	0.93
11	apple	172.0	7.1	7.6	0.92
12	apple	154.0	7.0	7.1	0.88
13	apple	164.0	7.3	7.7	0.70
14	apple	152.0	7.6	7.3	0.69
15	apple	156.0	7.7	7.1	0.69
16	apple	1000.0	7.6	7.5	0.67
17	lemon	NaN	7.2	NaN	0.70
18	lemon	NaN	7.3	NaN	0.72

19	lemon	NaN	7.2	NaN	0.72
20	lemon	NaN	7.3	NaN	0.71
21	lemon	NaN	7.3	NaN	0.72
22	lemon	NaN	7.3	NaN	0.72
23	lemon	NaN	5.8	NaN	0.73
24	lemon	NaN	6.0	NaN	0.71
25	orange	NaN	9.0	NaN	0.75
26	orange	356.0	9.2	9.2	0.75
27	orange	362.0	9.6	9.2	0.74
28	orange	204.0	7.5	9.2	0.77
29	orange	140.0	6.7	7.1	0.72
30	orange	160.0	7.0	7.4	0.81
31	orange	158.0	7.1	7.5	0.79
32	orange	210.0	7.8	8.0	0.82
33	orange	164.0	7.2	7.0	0.80
34	orange	190.0	7.5	8.1	0.74
35	orange	142.0	7.6	7.8	0.75
36	orange	150.0	7.1	7.9	0.75
37	orange	160.0	7.1	7.6	0.76
38	orange	154.0	7.3	7.3	0.79
39	orange	158.0	7.2	7.8	0.77
40	orange	154.0	7.3	7.5	0.76
41	orange	30000.0	7.1	7.5	0.78
42	orange	180.0	7.6	8.2	0.79
43	orange	154.0	7.2	7.2	0.82
44	lemon	194.0	7.2	10.3	0.70
45	lemon	200.0	7.3	10.5	0.72
46	lemon	186.0	7.2	9.2	0.72
47	lemon	216.0	7.3	10.2	0.71
48	lemon	196.0	7.3	10.2	0.72
49	lemon	174.0	7.3	10.1	0.72
50	lemon	132.0	5.8	8.7	0.73
51	lemon	130.0	6.0	8.2	0.71
52	lemon	116.0	6.0	7.5	0.72
53	lemon	118.0	5.9	8.0	0.72
54	lemon	120.0	6.0	8.4	0.74
55	lemon	116.0	6.1	8.5	0.71
56	lemon	116.0	6.3	7.7	0.72
57	lemon	116.0	5.9	8.1	0.73
58	lemon	152.0	6.5	8.5	0.72
59	lemon	118.0	6.1	8.1	0.70

5 You can use mean value to replace the missing values in case the data distribution is symmetric.

[4]:
```
import pandas as pd
import numpy as np
from scipy.stats import skew

# Function to check Data distribution is symmetric
# checking those columns which have Nan in it
def is_symmetric(data):
    s = skew(data)
    if abs(s) < 0.5:
        print("The data is symmetrically distributed")
    else:
        print("The data is not symmetrically distributed")

is_symmetric(df['height'].to_numpy())
is_symmetric(df['mass'].to_numpy())
```

```
The data is not symmetrically distributed
The data is not symmetrically distributed
```

6 For Both the columns that have NAN(missing values) is not symmetric so no need to replace with mean values.

7 d. Divide the data into training and testing sets, with the first 50 rows being used for training and the remaining 10 rows being used for testing.

[ ]:
```

```

[5]:
```
from sklearn.model_selection import train_test_split
df.replace(np.nan,0)
X = df[['mass','width','height','color_score']]
Y = df['fruit_name']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,␣
↪random_state = 0)
```

[6]:
```
X_train.head()
```

[6]:

	mass	width	height	color_score
30	160.0	7.0	7.4	0.81
41	30000.0	7.1	7.5	0.78
33	164.0	7.2	7.0	0.80
43	154.0	7.2	7.2	0.82

```
49    174.0    7.3    10.1          0.72
```

[7]:
```
X_test.head()
```

[7]:

	mass	width	height	color_score
26	356.0	9.2	9.2	0.75
35	142.0	7.6	7.8	0.75
59	118.0	6.1	8.1	0.70
28	204.0	7.5	9.2	0.77
11	172.0	7.1	7.6	0.92

[8]:
```
y_train.head()
```

[8]:
```
30    orange
41    orange
33    orange
43    orange
49     lemon
Name: fruit_name, dtype: object
```

[9]:
```
y_test.head()
```

[9]:
```
26    orange
35    orange
59     lemon
28    orange
11     apple
Name: fruit_name, dtype: object
```

[10]:
```
X_train.shape
```

[10]: (48, 4)

[11]:
```
X_test.shape
```

[11]: (12, 4)

[12]:
```
y_test.shape
```

[12]: (12,)

[13]:
```
y_train.shape
```

[13]: (48,)

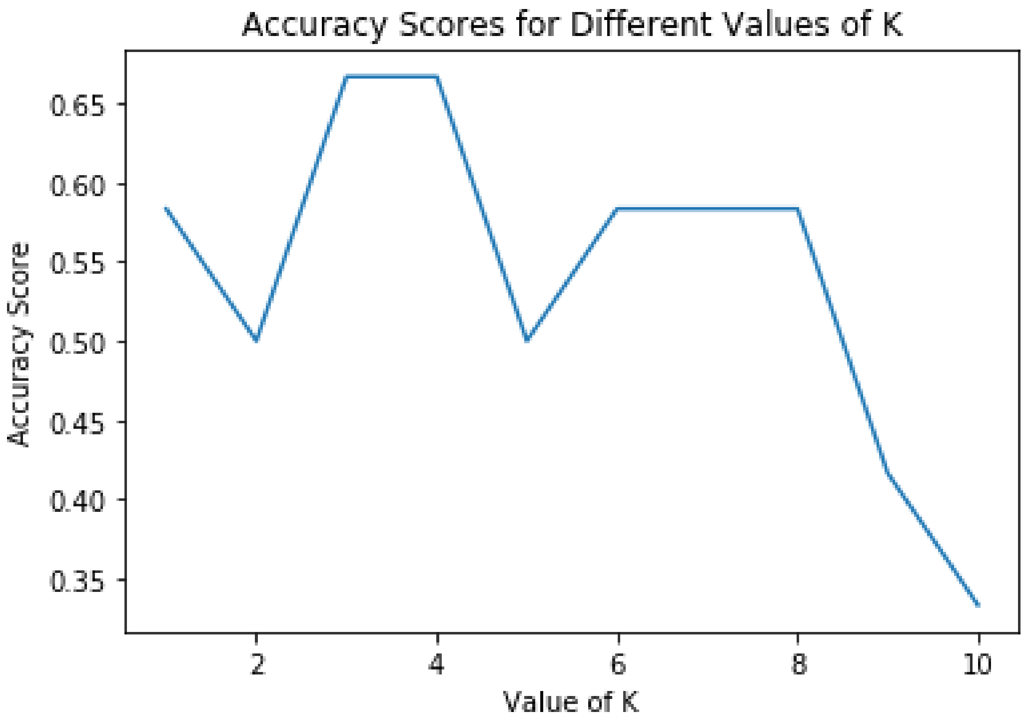
8 Apply the KNN model for different values of K (ranging from 1 to 10) and examine the results.

[14]:
```
kvalues_list = list(range(1, 11))
accuracy_scores = list(range(1, 11))
X_train = np.array(X_train.values.tolist())
y_train = np.array(y_train.values.tolist())
X_test = np.array(X_test.values.tolist())
y_test = np.array(y_test.values.tolist())

for i in kvalues_list:
    knntest_fruit_classifier = KNN(k = i)

    knntest_fruit_classifier.fit(X_train, y_train)
    y_pred = knntest_fruit_classifier.predict(X_test)
    accuracy = np.mean(y_pred == y_test)
    print("For K = ", i , " Accuracy is = ", accuracy)
    accuracy_scores[i - 1] = accuracy
```

```
For K =  1  Accuracy is =  0.5833333333333334
For K =  2  Accuracy is =  0.5
For K =  3  Accuracy is =  0.6666666666666666
For K =  4  Accuracy is =  0.6666666666666666
For K =  5  Accuracy is =  0.5
For K =  6  Accuracy is =  0.5833333333333334
For K =  7  Accuracy is =  0.5833333333333334
For K =  8  Accuracy is =  0.5833333333333334
For K =  9  Accuracy is =  0.4166666666666667
For K =  10  Accuracy is =  0.3333333333333333
```

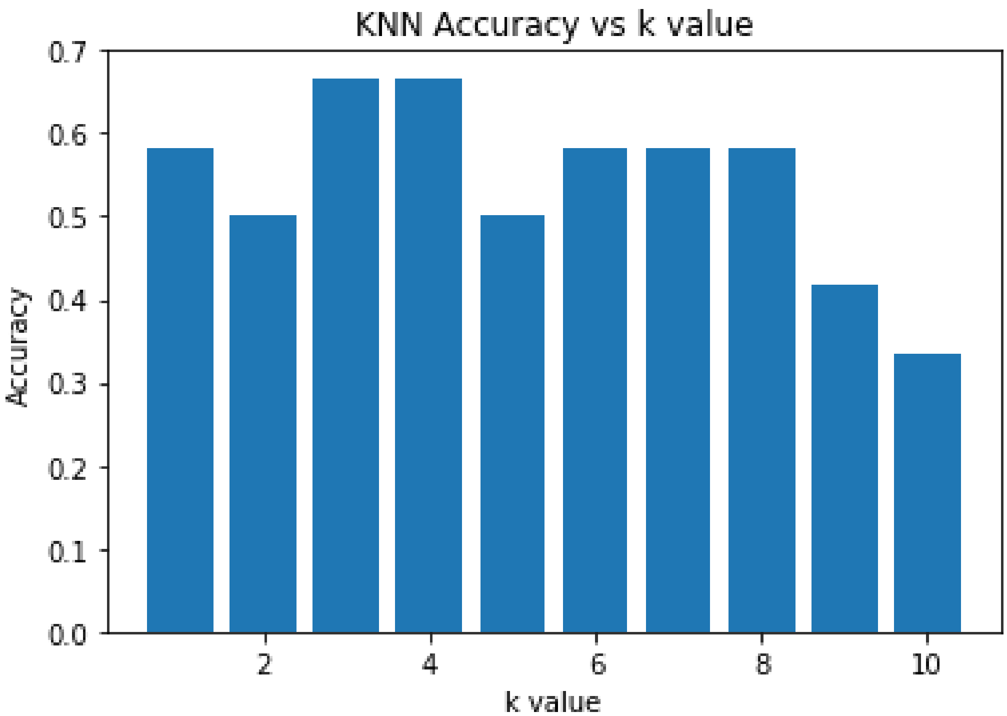
9 f. Plot the accuracy score for each value of K, to visualize the differences.

[15]:
```
plt.plot(kvalues_list, accuracy_scores)
plt.xlabel('Value of K')
plt.ylabel('Accuracy Score')
plt.title('Accuracy Scores for Different Values of K')
plt.show()
```



10 Bar Graph

[16]:
```
import matplotlib.pyplot as plt

plt.bar(kvalues_list, accuracy_scores)
plt.xlabel('k value')
plt.ylabel('Accuracy')
plt.title('KNN Accuracy vs k value')
plt.show()
```



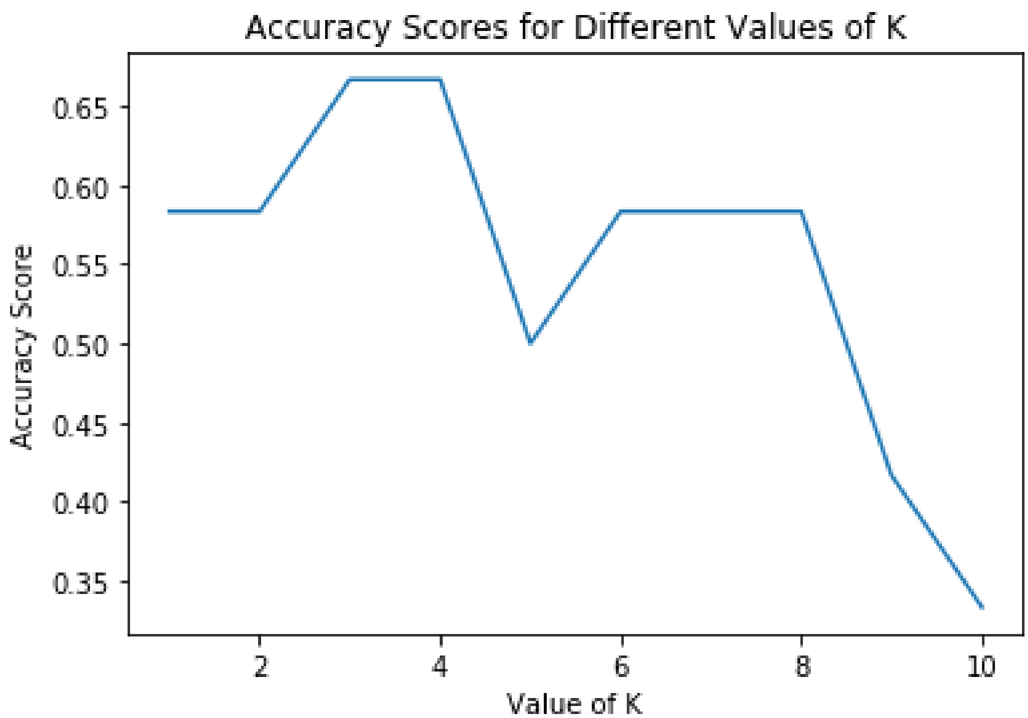
11 Using Manhattan Distance

[17]:
```
for i in kvalues_list:
    knntest_fruit_classifier_manhattan_distance = KNN(k = i)

    knntest_fruit_classifier_manhattan_distance.fit(X_train, y_train)
    y_pred = knntest_fruit_classifier_manhattan_distance.predict(X_test, 'man')
    accuracy = np.mean(y_pred == y_test)
    print("For K = ", i , " Accuracy is = ", accuracy)
    accuracy_scores[i - 1] = accuracy
```

```
For K =  1  Accuracy is =  0.5833333333333334
For K =  2  Accuracy is =  0.5833333333333334
For K =  3  Accuracy is =  0.6666666666666666
For K =  4  Accuracy is =  0.6666666666666666
For K =  5  Accuracy is =  0.5
For K =  6  Accuracy is =  0.5833333333333334
For K =  7  Accuracy is =  0.5833333333333334
For K =  8  Accuracy is =  0.5833333333333334
For K =  9  Accuracy is =  0.4166666666666667
For K =  10  Accuracy is =  0.3333333333333333
```

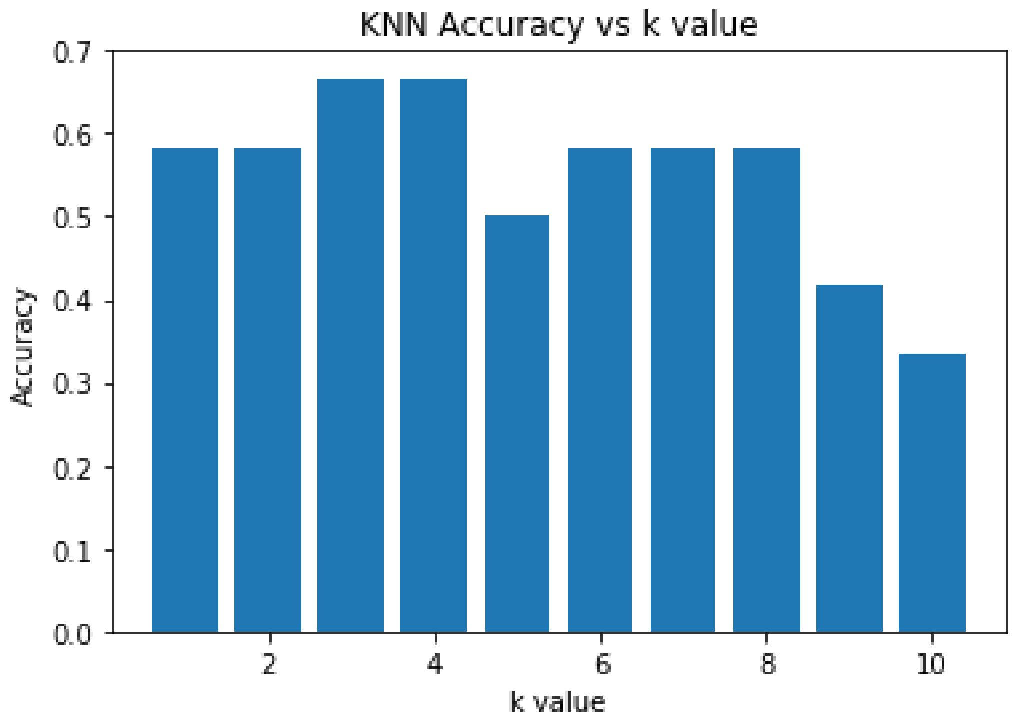
12 f. Plot the accuracy score for each value of K, to visualize the differences.

[18]:
```
plt.plot(kvalues_list, accuracy_scores)
plt.xlabel('Value of K')
plt.ylabel('Accuracy Score')
plt.title('Accuracy Scores for Different Values of K')
plt.show()
```



13 Bar Graph to visualize the differences of Manhattan Distance

[19]:
```
import matplotlib.pyplot as plt

plt.bar(kvalues_list, accuracy_scores)
plt.xlabel('k value')
plt.ylabel('Accuracy')
plt.title('KNN Accuracy vs k value')
plt.show()
```


KNN Accuracy vs k value
Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
2
4
6
8
10
k value