kNN on Iris Dataset ----Norina Akhtar

```
#importing the required libraries
import pandas as pd
import numpy as np
import operator
import matplotlib.pyplot as plt
```

```
# uplaod file from local drive
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving iris.csv to iris.csv
```

```
import io
data = pd.read_csv(io.BytesIO(uploaded['iris.csv']))
print(data)
```

```
     sepal.length  sepal.width  petal.length  petal.width    variety
0             5.1          3.5           1.4          0.2     Setosa
1             4.9          3.0           1.4          0.2     Setosa
2             4.7          3.2           1.3          0.2     Setosa
3             4.6          3.1           1.5          0.2     Setosa
4             5.0          3.6           1.4          0.2     Setosa
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  Virginica
146           6.3          2.5           5.0          1.9  Virginica
147           6.5          3.0           5.2          2.0  Virginica
148           6.2          3.4           5.4          2.3  Virginica
149           5.9          3.0           5.1          1.8  Virginica

[150 rows x 5 columns]
```

part a

Dividing the dataset as development and test.

```
#randomize the indices
indices = np.random.permutation(data.shape[0])
div = int(0.75 * len(indices))
development_id, test_id = indices[:div], indices[div:]
#dividing the dataset using randomized indices
development_set, test_set = data.loc[development_id,:], data.loc[test_id,:]
print("Development Set:\n", development_set, "\n\nTest Set:\n", test_set)
mean_development_set = development_set.mean(numeric_only=True)
mean_test_set = test_set.mean(numeric_only=True)
std_development_set = development_set.std(numeric_only=True)
std_test_set = test_set.std(numeric_only=True)
```

```
Development Set:
     sepal.length  sepal.width  petal.length  petal.width     variety
66           5.6          3.0           4.5          1.5  Versicolor
75           6.6          3.0           4.4          1.4  Versicolor
39           5.1          3.4           1.5          0.2      Setosa
34           4.9          3.1           1.5          0.2      Setosa
95           5.7          3.0           4.2          1.2  Versicolor
..           ...          ...           ...          ...         ...
83           6.0          2.7           5.1          1.6  Versicolor
9            4.9          3.1           1.5          0.1      Setosa
40           5.0          3.5           1.3          0.3      Setosa
60           5.0          2.0           3.5          1.0  Versicolor
7            5.0          3.4           1.5          0.2      Setosa

[112 rows x 5 columns]

Test Set:
      sepal.length  sepal.width  petal.length  petal.width     variety
29            4.7          3.2           1.6          0.2      Setosa
3             4.6          3.1           1.5          0.2      Setosa
91            6.1          3.0           4.6          1.4  Versicolor
109           7.2          3.6           6.1          2.5   Virginica
44            5.1          3.8           1.9          0.4      Setosa
137           6.4          3.1           5.5          1.8   Virginica
112           6.8          3.0           5.5          2.1   Virginica
```

38	4.4	3.0	1.3	0.2	Setosa
10	5.4	3.7	1.5	0.2	Setosa
24	4.8	3.4	1.9	0.2	Setosa
35	5.0	3.2	1.2	0.2	Setosa
94	5.6	2.7	4.2	1.3	Versicolor
100	6.3	3.3	6.0	2.5	Virginica
86	6.7	3.1	4.7	1.5	Versicolor
0	5.1	3.5	1.4	0.2	Setosa
61	5.9	3.0	4.2	1.5	Versicolor
88	5.6	3.0	4.1	1.3	Versicolor
106	4.9	2.5	4.5	1.7	Virginica
143	6.8	3.2	5.9	2.3	Virginica
67	5.8	2.7	4.1	1.0	Versicolor
56	6.3	3.3	4.7	1.6	Versicolor
36	5.5	3.5	1.3	0.2	Setosa
133	6.3	2.8	5.1	1.5	Virginica
121	5.6	2.8	4.9	2.0	Virginica
76	6.8	2.8	4.8	1.4	Versicolor
53	5.5	2.3	4.0	1.3	Versicolor
71	6.1	2.8	4.0	1.3	Versicolor
89	5.5	2.5	4.0	1.3	Versicolor
30	4.8	3.1	1.6	0.2	Setosa
113	5.7	2.5	5.0	2.0	Virginica
17	5.1	3.5	1.4	0.3	Setosa
12	4.8	3.0	1.4	0.1	Setosa
14	5.8	4.0	1.2	0.2	Setosa
114	5.8	2.8	5.1	2.4	Virginica
70	5.9	3.2	4.8	1.8	Versicolor
42	4.4	3.2	1.3	0.2	Setosa
136	6.3	3.4	5.6	2.4	Virginica
31	5.4	3.4	1.5	0.4	Setosa

Part b

Implement kNN using the following hyperparameters:

number of neighbor

```
* 1,3,5,7
```

distance metric

```
* euclidean distance
* normalized euclidean distance
* cosine similarity
```

Retrieving the 'class' column from the development and test sets and storing it in separate lists. Calculating the mean and standard deviation of the development set and test set for normalizing the data.

```
test_class = list(test_set.iloc[:,-1])
dev_class = list(development_set.iloc[:,-1])
mean_development_set = development_set.mean(numeric_only=True)
mean_test_set = test_set.mean(numeric_only=True)
std_development_set = development_set.std(numeric_only=True)
std_test_set = test_set.std(numeric_only=True)
```

Functions for computing the Euclidean Distance, Normalized Euclidean Distance, Cosine Similarity and k Nearest Neighbor to determine the 'class' for a given input instance.

```
def euclideanDistance(data_1, data_2, data_len):
    dist = 0
    for i in range(data_len):
        dist = dist + np.square(data_1[i] - data_2[i])
    return np.sqrt(dist)

def normalizedEuclideanDistance(data_1, data_2, data_len, data_mean, data_std):
    n_dist = 0
    for i in range(data_len):
        n_dist = n_dist + (np.square(((data_1[i] - data_mean[i])/data_std[i]) - ((data_2[i] - data_mean[i])/data_std[i])))
    return np.sqrt(n_dist)

```

```
def cosineSimilarity(data_1, data_2):
    dot = np.dot(data_1, data_2[:-1])
    norm_data_1 = np.linalg.norm(data_1)
    norm_data_2 = np.linalg.norm(data_2[:-1])
    cos = dot / (norm_data_1 * norm_data_2)
    return (1-cos)

def knn(dataset, testInstance, k, dist_method, dataset_mean, dataset_std):
    distances = {}
    length = testInstance.shape[1]
    if dist_method == 'euclidean':
        for x in range(len(dataset)):
            dist_up = euclideanDistance(testInstance, dataset.iloc[x], length)
            distances[x] = dist_up[0]
    elif dist_method == 'normalized_euclidean':
        for x in range(len(dataset)):
            dist_up = normalizedEuclideanDistance(testInstance, dataset.iloc[x], length, dataset_mean, dataset_std)
            distances[x] = dist_up[0]
    elif dist_method == 'cosine':
        for x in range(len(dataset)):
            dist_up = cosineSimilarity(testInstance, dataset.iloc[x])
            distances[x] = dist_up[0]
 # Sort values based on distance
    sort_distances = sorted(distances.items(), key=operator.itemgetter(1))
    neighbors = []
    # Extracting nearest k neighbors
    for x in range(k):
        neighbors.append(sort_distances[x][0])
    # Initializing counts for 'class' labels counts as 0
    counts = {"Iris-setosa" : 0, "Iris-versicolor" : 0, "Iris-virginica" : 0}
    # Computing the most frequent class
    for x in range(len(neighbors)):
        response = dataset.iloc[neighbors[x]][-1]
        if response in counts:
            counts[response] += 1
        else:
            counts[response] = 1
    # Sorting the class in reverse order to get the most frequest class
    sort_counts = sorted(counts.items(), key=operator.itemgetter(1), reverse=True)
    return(sort_counts[0][0])
```

Part c

Using the development data set

Iterating all of the development data points and computing the class for each k and each distance metric

```
print(development_set)

```

```
    sepal.length  sepal.width  petal.length  petal.width     variety
66           5.6          3.0           4.5          1.5  Versicolor
75           6.6          3.0           4.4          1.4  Versicolor
39           5.1          3.4           1.5          0.2      Setosa
34           4.9          3.1           1.5          0.2      Setosa
95           5.7          3.0           4.2          1.2  Versicolor
..           ...          ...           ...          ...         ...
83           6.0          2.7           5.1          1.6  Versicolor
9            4.9          3.1           1.5          0.1      Setosa
40           5.0          3.5           1.3          0.3      Setosa
60           5.0          2.0           3.5          1.0  Versicolor
7            5.0          3.4           1.5          0.2      Setosa

[112 rows x 5 columns]
```

```
# Creating a list of list of all columns except 'class' by iterating through the development set
row_list = []
for index, rows in development_set.iterrows():
  # print()

  # row_list.append(rows['sepal.length'])
   my_list =[rows["sepal.length"], rows["sepal.width"], rows["petal.length"], rows["petal.width"]]
   row_list.append([my_list])
# k values for the number of neighbors that need to be considered
k_n = [1, 3, 5, 7]
# Distance metrics
```

```
distance_methods = ['euclidean', 'normalized_euclidean', 'cosine']
# Performing kNN on the development set by iterating all of the development set data points and for each k and each distance
obs_k = {}
for dist_method in distance_methods:
    development_set_obs_k = {}
    for k in k_n:
        development_set_obs = []
        for i in range(len(row_list)):
            development_set_obs.append(knn(development_set, pd.DataFrame(row_list[i]), k, dist_method, mean_development_set,
        development_set_obs_k[k] = development_set_obs
    # Nested Dictionary containing the observed class for each k and each distance metric (obs_k of the form obs_k[dist_methc
    obs_k[dist_method] = development_set_obs_k
    print(dist_method.upper() + " distance method performed on the dataset for all k values!")
```

```
EUCLIDEAN distance method performed on the dataset for all k values!
NORMALIZED_EUCLIDEAN distance method performed on the dataset for all k values!
COSINE distance method performed on the dataset for all k values!
```

Computing the accuracy for the development data set and finding the optimal hyperparametes

```
# Calculating the accuracy of the development set by comparing it with the development set 'class' list created earlier
accuracy = {}
for key in obs_k.keys():
    accuracy[key] = {}
    for k_value in obs_k[key].keys():
        #print('k = ', key)
        count = 0
        for i,j in zip(dev_class, obs_k[key][k_value]):
            if i == j:
                count = count + 1
            else:
                pass
        accuracy[key][k_value] = count/(len(dev_class))

# Storing the accuracy for each k and each distance metric into a dataframe
df_res = pd.DataFrame({'k': k_n})
for key in accuracy.keys():
    value = list(accuracy[key].values())
    df_res[key] = value
print(df_res)

# Plotting a Bar Chart for accuracy
draw = df_res.plot(x='k', y=['euclidean', 'normalized_euclidean', 'cosine'], kind="bar", colormap='YlGnBu')
draw.set(ylabel='Accuracy')

# Ignoring k=1 if the value of accuracy for k=1 is 100%, since this mostly implies overfitting
df_res.loc[df_res['k'] == 1.0, ['euclidean', 'normalized_euclidean', 'cosine']] = np.nan

# Fetching the best k value for using all hyper-parameters
# In case the accuracy is the same for different k and different distance metric selecting the first of all the same
column_val = [c for c in df_res.columns if not c.startswith('k')]
col_max = df_res[column_val].max().idxmax()
best_dist_method = col_max
row_max = df_res[col_max].argmax()
best_k = int(df_res.iloc[row_max]['k'])
if df_res.isnull().values.any():
    print('\n\n\nBest k value is\033[1m', best_k, '\033[0mand best distance metric is\033[1m', best_dist_method, '\033[0m. Ig
else:
    print('\n\n\nBest k value is\033[1m', best_k, '\033[0mand best distance metric is\033[1m', best_dist_method, '\033[0m.')
```

	k	euclidean	normalized_euclidean	cosine
0	1	1.000000	1.000000	1.000000
1	3	0.973214	0.964286	0.973214
2	5	0.964286	0.964286	0.964286
3	7	0.973214	0.973214	0.973214

Using the test dataset

```
print('\n\n\nBest k value is\033[1m', best_k, '\033[0mand best distance metric is\033[1m', best_dist_method, '\033[0m')
```

```
Best k value is 3 and best distance metric is euclidean
```

Using the best k value and best distance metric to determine the class for all rows in the test dataset

```
# Creating a list of list of all columns except 'class' by iterating through the development set
row_list_test = []
for index, rows in test_set.iterrows():
    my_list =[rows["sepal.length"], rows["sepal.width"], rows["petal.length"], rows["petal.width"]]
    row_list_test.append([my_list])
test_set_obs = []
for i in range(len(row_list_test)):
    test_set_obs.append(knn(test_set, pd.DataFrame(row_list_test[i]), best_k, best_dist_method, mean_test_set, std_test_set))
#print(test_set_obs)

count = 0
for i,j in zip(test_class, test_set_obs):
    if i == j:
        count = count + 1
    else:
        pass
accuracy_test = count/(len(test_class))
print('Final Accuracy of the Test dataset is ', accuracy_test)
```

```
Final Accuracy of the Test dataset is  0.9473684210526315
```

References

https://hc.labnet.sfbu.edu/~henry/sfbu/course//machine_learning_with_r_cookbook/practice_ml_with_r/slide/iris.html

https://ai.plainenglish.io/understanding-confusion-matrix-and-applying-it-on-knn-classifier-on-iris-dataset-b57f85d05cd8

https://stackoverflow.com/questions/16476924/how-to-iterate-over-rows-in-a-dataframe-in-pandas