**Ademilton Marcelo da Cruz Nunes (19679)**

**CS550 - Machine Learning and Business Intelligence**

**Week 4 - Homework 1**

## Q30. KNN + Confusion Matrix + Iris Data set + Colab

kNNiris.ipynb file changes:
Changed reading data to read a local CSV file.
Commented parts of the code to understanding
Saved on Github:
https://github.com/ademiltonnunes/Machine-Learning/tree/main/Supervised%20Learning/KNN%20%2B%20Confusion%20Matrix%20%2B%20Iris%20Data%20set%20%2B%20Colab

# kNN on Iris Dataset

### Author - Ishita Kapur

In [24]:

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

In [25]:

```
#reading data from the csv file

# data = pd.read_csv('iris.data', header=None, names=['sepal_length', 'sepal_width', 'p
etal_length', 'petal_width', 'class'])
# print(data)

# getting data from .cvs file locally
from google.colab import files
uploaded  = files.upload()
import io
data = pd.read_csv(
    io.BytesIO(uploaded['iris.csv']),
    names=["sepal_length", "sepal_width", "petal_length", "petal_width", "class"])
print(data)
```

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 (1).csv
     sepal_length  sepal_width  petal_length  petal_width        class
0             5.1          3.5           1.4          0.2     I.setosa
1             4.9          3.0           1.4          0.2     I.setosa
2             4.7          3.2           1.3          0.2     I.setosa
3             4.6          3.1           1.5          0.2     I.setosa
4             5.0          3.6           1.4          0.2     I.setosa
..            ...          ...           ...          ...          ...
145           6.7          3.0           5.2          2.3  I.virginica
146           6.3          2.5           5.0          1.9  I.virginica
147           6.5          3.0           5.2          2.0  I.virginica
148           6.2          3.4           5.4          2.3  I.virginica
149           5.9          3.0           5.1          1.8  I.virginica

[150 rows x 5 columns]
```

## Part a)

Dividing the dataset as development and test.

In [26]:

```
#randomize the all indices
indices = np.random.permutation(data.shape[0])

#getting 75% of the random indices
div = int(0.75 * len(indices))

#development_id indice 0-111 = 112
#test_id indices 112-150 = 38
development_id, test_id = indices[:div], indices[div:]

#dividing the dataset using randomized indices
#112 indices for Development and 38 for test
#bellow is getting the complete object of each index
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()
# mean_test_set = test_set.mean()
# std_development_set = development_set.std()
# std_test_set = test_set.std()
```

```
Development Set:
     sepal_length  sepal_width  petal_length  petal_width         class
79            5.7          2.6           3.5          1.0  I.versicolor
63            6.1          2.9           4.7          1.4  I.versicolor
104           6.5          3.0           5.8          2.2   I.virginica
108           6.7          2.5           5.8          1.8   I.virginica
91            6.1          3.0           4.6          1.4  I.versicolor
..            ...          ...           ...          ...           ...
61            5.9          3.0           4.2          1.5  I.versicolor
101           5.8          2.7           5.1          1.9   I.virginica
35            5.0          3.2           1.2          0.2      I.setosa
114           5.8          2.8           5.1          2.4   I.virginica
52            6.9          3.1           4.9          1.5  I.versicolor

[112 rows x 5 columns]

Test Set:
     sepal_length  sepal_width  petal_length  petal_width         class
126           6.2          2.8           4.8          1.8   I.virginica
42            4.4          3.2           1.3          0.2      I.setosa
148           6.2          3.4           5.4          2.3   I.virginica
99            5.7          2.8           4.1          1.3  I.versicolor
112           6.8          3.0           5.5          2.1   I.virginica
26            5.0          3.4           1.6          0.4      I.setosa
137           6.4          3.1           5.5          1.8   I.virginica
134           6.1          2.6           5.6          1.4   I.virginica
34            4.9          3.1           1.5          0.2      I.setosa
25            5.0          3.0           1.6          0.2      I.setosa
85            6.0          3.4           4.5          1.6  I.versicolor
58            6.6          2.9           4.6          1.3  I.versicolor
51            6.4          3.2           4.5          1.5  I.versicolor
68            6.2          2.2           4.5          1.5  I.versicolor
82            5.8          2.7           3.9          1.2  I.versicolor
40            5.0          3.5           1.3          0.3      I.setosa
95            5.7          3.0           4.2          1.2  I.versicolor
93            5.0          2.3           3.3          1.0  I.versicolor
123           6.3          2.7           4.9          1.8   I.virginica
37            4.9          3.6           1.4          0.1      I.setosa
30            4.8          3.1           1.6          0.2      I.setosa
87            6.3          2.3           4.4          1.3  I.versicolor
120           6.9          3.2           5.7          2.3   I.virginica
107           7.3          2.9           6.3          1.8   I.virginica
78            6.0          2.9           4.5          1.5  I.versicolor
9             4.9          3.1           1.5          0.1      I.setosa
41            4.5          2.3           1.3          0.3      I.setosa
83            6.0          2.7           5.1          1.6  I.versicolor
28            5.2          3.4           1.4          0.2      I.setosa
113           5.7          2.5           5.0          2.0   I.virginica
6             4.6          3.4           1.4          0.3      I.setosa
115           6.4          3.2           5.3          2.3   I.virginica
77            6.7          3.0           5.0          1.7  I.versicolor
4             5.0          3.6           1.4          0.2      I.setosa
55            5.7          2.8           4.5          1.3  I.versicolor
139           6.9          3.1           5.4          2.1   I.virginica
0             5.1          3.5           1.4          0.2      I.setosa
70            5.9          3.2           4.8          1.8  I.versicolor
```

## 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.

In [27]:

```
#Getting only the label/class of each element in the test and dev set
test_class = list(test_set.iloc[:,-1])
dev_class = list(development_set.iloc[:,-1])

#Getting the mean of dev and test set
mean_development_set = development_set.mean()
mean_test_set = test_set.mean()

#Standard Deviation of dev and test set, if the value is higher, far from mean. If valu
e is low, close to mean
std_development_set = development_set.std()
std_test_set = test_set.std()
```

```
<ipython-input-27-641a240cbf9e>:6: FutureWarning: Dropping of nuisance col
umns in DataFrame reductions (with 'numeric_only=None') is deprecated; in
a future version this will raise TypeError.  Select only valid columns bef
ore calling the reduction.
  mean_development_set = development_set.mean()
<ipython-input-27-641a240cbf9e>:7: FutureWarning: Dropping of nuisance col
umns in DataFrame reductions (with 'numeric_only=None') is deprecated; in
a future version this will raise TypeError.  Select only valid columns bef
ore calling the reduction.
  mean_test_set = test_set.mean()
<ipython-input-27-641a240cbf9e>:10: FutureWarning: Dropping of nuisance co
lumns in DataFrame reductions (with 'numeric_only=None') is deprecated; in
a future version this will raise TypeError.  Select only valid columns bef
ore calling the reduction.
  std_development_set = development_set.std()
<ipython-input-27-641a240cbf9e>:11: FutureWarning: Dropping of nuisance co
lumns in DataFrame reductions (with 'numeric_only=None') is deprecated; in
a future version this will raise TypeError.  Select only valid columns bef
ore calling the reduction.
  std_test_set = test_set.std()
```

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

In [28]:

```
#3 different techniques to define the distance of each neighbor

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)

# Title: Cosine Similarty between 2 Number Lists
# Author: dontloo
# Date: 03.27.2017
# Code version: 1
# Availability: https://stackoverflow.com/questions/18424228/cosine-similarity-between-
2-number-lists
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], lengt
h, 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

In [29]:

```
# Creating a list of list of all columns except 'class' by iterating through the develo
pment set
row_list = []
for index, rows in development_set.iterrows():
    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 po
ints and for each k and each distance metric
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, std_development_set))
        development_set_obs_k[k] = development_set_obs
    # Nested Dictionary containing the observed class for each k and each distance metr
ic (obs_k of the form obs_k[dist_method][k])
    obs_k[dist_method] = development_set_obs_k
    print(dist_method.upper() + " distance method performed on the dataset for all k va
lues!")
# print(obs_k)
```

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

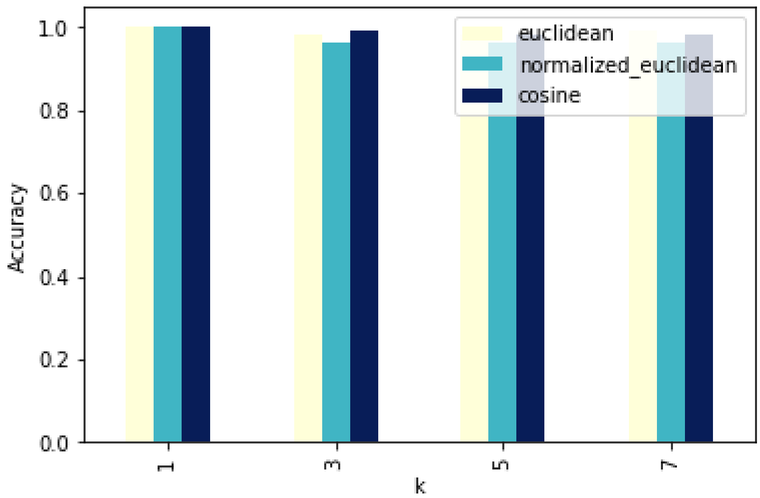
Computing the accuracy for the development data set and finding the optimal hyperparametes

In [30]:

```
# 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="ba
r", colormap='YlGnBu')
draw.set(ylabel='Accuracy')

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

# Fetching the best k value for using all hyper-parameters
# In case the accuracy is the same for different k and different distance metric select
ing 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\0
33[1m', best_dist_method, '\033[0m. Ignoring k=1 if the value of accuracy for k=1 is 10
0%, since this mostly implies overfitting')
else:
    print('\n\n\nBest k value is\033[1m', best_k, '\033[0mand best distance metric is\0
33[1m', best_dist_method, '\033[0m.')
```

```
   k  euclidean  normalized_euclidean    cosine
0  1   1.000000              1.000000  1.000000
1  3   0.982143              0.964286  0.991071
2  5   0.982143              0.964286  0.982143
3  7   0.991071              0.964286  0.982143
```

Best k value is **7** and best distance metric is **euclidean** . Ignoring k=1 if the value of accuracy for k=1 is 100%, since this mostly implies overfitting




## Part d)

Using the test dataset

In [31]:

```
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 **7** 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

In [32]:

```
# Creating a list of list of all columns except 'class' by iterating through the develo
pment 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  0.868421052631579
```

## References

https://stackoverflow.com/questions/18424228/cosine-similarity-between-2-number-lists (https://stackoverflow.com/questions/18424228/cosine-similarity-between-2-number-lists) - for cosine similarity

https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/ (https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/) - for nearest neighbors

In [33]:

```
! jupyter nbconvert --to html kNN_iris.ipynb
```

```
[NbConvertApp] Converting notebook kNN_iris.ipynb to html
[NbConvertApp] Writing 369790 bytes to kNN_iris.html
```