Mandatory exercise No.2: K-Nearest Neighbors

In this exercise, you are going to implement the K-Nearest Neighbors algorithm from scratch in python. We use the Iris Flower Species dataset from UCI machine learning repository (https://archive.ics.uci.edu/ml/datasets/Iris).

Objective of the exercise:

- How to implement K-Nearest Neighbors algorithm in python
- How to work with real dataset and evaluate KNN algorithm
- How to use KNN to make a prediction for new data

Dataset Description:

This dataset has been used to predict the flower species utilizing the measurements of the Iris flowers. Iris is a plant with showy flowers. It takes its name from the Greek word for a rainbow, which is also the name for the Greek goddess of the rainbow, Iris [Wikipedia].

The dataset contains 150 observations (rows) with 4 features (input variables) and one output variable (label). The Iris dataset is a balanced dataset, which means that there are equal number of instances in each class. The variable names are as follow:

- Sepal length in cm.
- Sepal width in cm.
- Petal length in cm.
- Petal width in cm.
- Class
  - Iris Setosa
  - Iris Versicolour
  - Iris Virginica

To read the data, we use python library Pandas. Pandas is a powerful and open source library for data analysis, data manipulation, and etc. For more details, check out the Pandas documentation page (https://pandas.pydata.org/docs/).

In [1]:
```
import pandas as pd
```

In [2]:
```
# Iris dataset link
URL = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'

df_iris = pd.read_csv(URL,
                      header=None,
                      names=['sepal length', 'sepal width',
                             'petal length', 'petal width', 'class'])
```

Below we can see the first five rows of Iris dataset.

In [3]:
```
df_iris.head(5)
```

Out[3]:

	sepal length	sepal width	petal length	petal width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

As you can see, the class values are string. Indeed, for the calculations, we need to assign a label to each class.

One way to achieve this is to use the $factorize()$ function from Pandas. This function returns a tuple of an integer array of labels and an array of unique class values. See the Pandas documentation for more details (https://pandas.pydata.org/docs/reference/api/pandas.factorize.html).

In [4]:
```
pd.factorize(df_iris['class'])
```

Out[4]:
```
(array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 Index(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype='object'))
```

Task:

For the KNN algorithm, you need to implement three main functions to:

1. Calculate the distance
2. Find the nearest neighbors
3. Make the predictions

For the implementation, you need to have python3 environment.

Distance Calculation

To calculate the distance, we use the Euclidean distance. It formalized as below:

$$d(p,q) = \sqrt{\sum_{i=1}^{N}(q_i - p_i)^2}$$

Where p and q are two feature vectors from our dataset. i is the index of a specific feature and N is the number of features in the dataset.

With Euclidean distance, the smaller values represent the higher similarity between corresponding feature vctors. The value zero indicates the absolute similarity.

Use sum, exp and $sqrt$ from python to implement this function. Check the Python3 documentation for further details (https://docs.python.org/3/library/math.html).

Finding the Nearest Neighbors

To find the nearest neighbors, we need to calculate the distance between each data points in our dataset and the new data point. We can do so by using the distance function explained above.

After calculating the distances, we sort all of the data points based on their distance to the new data point. Then we return the top K as the nominated neighbors.

Make predictions

The new data point's class can be determined from the most similar neighbors that we have obtained. As we work on a classification task, our prediction will be the most represented class among the selected neighbors.

To achieve this, you can use the $max()$ function in python. Check the Python3 documentation for the details (https://docs.python.org/3/library/functions.html#max).

Goal

The goal of this exercise is to classify the data point below.

In [5]:
```
new_dp = [7.0, 3.1, 1.3, 0.7]
```

Assesment

50% Documentation. You should document your code where you explain how the algorithm works. The documentation can be in .pdf format or, alternatively, detailed comments in the code.

50% Implementation. Optimized and well-written code is beneficial.