Cairo University
Faculty of Computers and Artificial Intelligence
Machine Learning Course

Lab 5 - Part 2: K-Nearest Neighbors

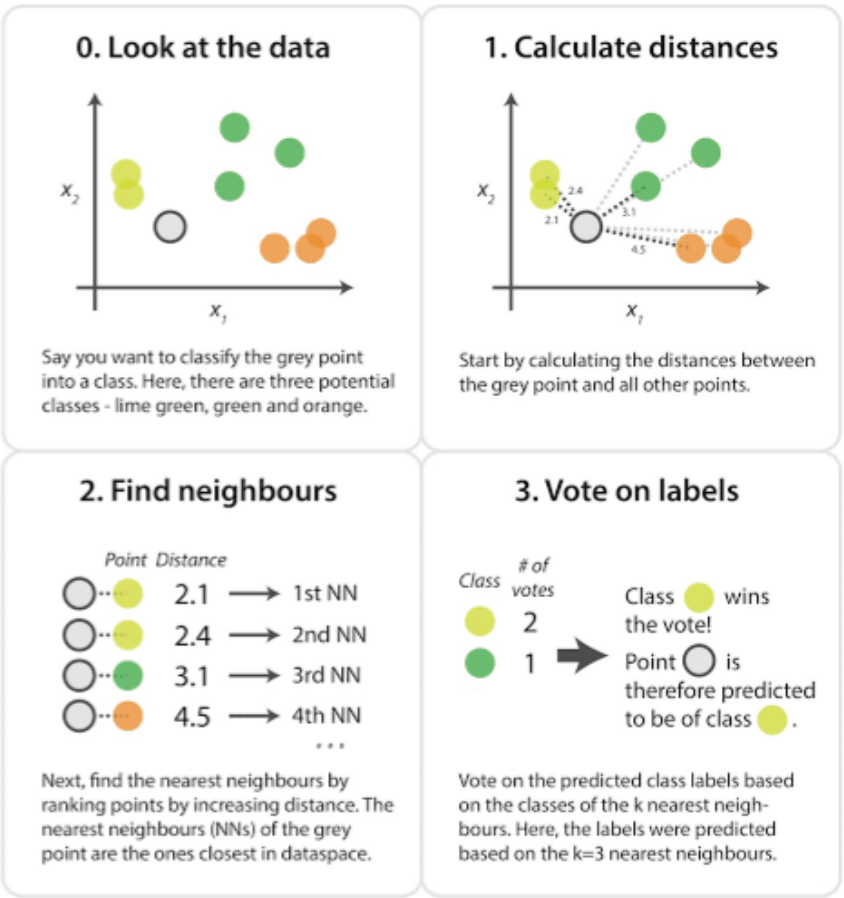
Lab Outline:

- Summary of k-nearest neighbors
- Steps of applying k-nearest neighbors to a dataset
- Practice

Summary of K-Nearest Neighbors:

K-nearest neighbors (KNN) is a ***supervised*** machine learning algorithm that is typically used for ***classification*** tasks (but can also be used for ***regression***). It is based on the assumption that similar points can be found near one another. KNN is ***non-parametric*** which means that it doesn't make any assumptions about the underlying distribution of the data.

KNN works by identifying the *k* nearest neighbors of a given query point and assigning the majority class in the nearest neighbors to that point.



The ***k value*** in KNN defines how many neighbors will be checked to determine the class of a specific query point. The ***choice of k*** will largely depend on the input data as data with more outliers or noise will likely perform better with higher values of *k*. It is also recommended to have an odd number for *k* to avoid ties in classification.

In order to determine which data points are the nearest neighbors of a given query point, the ***distance*** between the query point and the other data points needs to be calculated. Of course, there are several distance measures to choose from (e.g. Manhattan distance, Hamming distance, etc.). The most commonly used distance measure is the ***Euclidean distance*** which is calculated by the following formula:

$$(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

KNN can be implemented in different ways. The most naïve neighbor search implementation involves the ***brute-force*** computation of distances between the query point and all points in the dataset (which was explained earlier). Since the brute-force approach is computationally inefficient, a variety of tree-based data structures have been invented to reduce the required number of distance calculations. Two popular examples are the ***KD tree*** and the ***Ball tree*** data structures.

Notice that KNN is a ***lazy learner***; meaning that it only stores the training dataset in the training phase which also means that all the computation occurs when the classification is being made. In other words, it doesn't explicitly learn a model. Since KNN heavily relies on memory to store all its training data, it is also referred to as an ***instance-based*** or ***memory-based*** learning method. It is worth noting that the minimal training phase of KNN comes at both a memory cost as well as a computational cost (during classification).

However, one of the most attractive features of KNN is that it is simple to understand and easy to implement. Furthermore, KNN works just as easily with ***multiclass datasets*** whereas many other algorithms are hardcoded for the binary setting.

Steps of Applying K-Nearest Neighbors to a Dataset:

1. **Analyze** the dataset
2. Perform data **preprocessing** *(including **splitting** into train and test sets)*
3. **Choose *k*** and **store** the training data *(training phase)*
4. **Assess** the fitted model on the test data *(run KNN on the test data)*
5. Generate **predictions** for new data *(run KNN on the new data)*

Recall the example from the previous lab:

Suppose you are the administrator of a university department and you want to determine each applicant's chance of acceptance based on their results on two exams. You have data from previous applicants which consists of the applicants' scores on two exams and the admissions decision.

You can use KNN to get the class of the acceptance status based on the scores from those two exams.

Note: The code can be found inside "Lab 5.ipynb".

Practice:

I. You are given the heights, weights and T-shirt sizes of some customers in the table below and you need to predict the T-shirt size of a new customer given only his height and weight. Use KNN *(k=3)* to predict the T-shirt size of this new customer whose height and weight are 161 cm and 66 kg respectively.

Height (cm)	Weight (kg)	T-shirt Size
158	58	M
158	63	M
160	59	M
160	60	M
160	64	M
163	60	M
163	61	M
163	64	L
165	61	L
165	65	L
168	62	L
168	66	L
170	63	L
170	68	L

II. Write the pseudocode for KNN:

Function runKNN

Input: *k, point, data_x, data_y*
Output: *class*

1:
2:
3:
...

III. Could you provide a Python implementation of the K-Nearest Neighbors algorithm from scratch? The implementation should be applied on the Iris dataset (which can be loaded from the UCI Machine Learning repository). Additionally, the solution should include a method to calculate the accuracy of the model's predictions on a test set.

Given the following solution structure, implement the **"predict"** function.

```
import numpy as np
from collections import Counter
import pandas as pd
from sklearn.model_selection import train_test_split
from scipy.spatial import distance

# k-NN Classifier
class KNN:
    def __init__(self, k=3):
        self.k = k

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

    def predict(self, X):
       // to be implemented

# Load the Iris dataset from a URL
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
df = pd.read_csv(url, names=names)

# Split dataset into features and target
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15,
random_state=1)  # 85% training and 15% test

# Create a k-NN Classifier
knn = KNN(k=3)

# Train the model using the training set
knn.fit(X_train, y_train)

# Predict the response for test set
y_pred = knn.predict(X_test)

# Model Accuracy
print("Accuracy:", np.sum(y_pred == y_test) / len(y_test))
```