

TP4: K-NN

Réalisé par:

- El Ghoul Abdessalam

Master – S.I.R
2021/2022

TP 4 : K-NN

- Introduction :

En intelligence artificielle, plus précisément en apprentissage automatique, la **méthode des k plus proches voisins** est une méthode d'apprentissage supervisé. En abrégé k-NN ou KNN, de l'anglais *k-nearest neighbors*.

Dans ce cadre, on dispose d'une base de données d'apprentissage constituée de N couples « entrée-sortie ». Pour estimer la sortie associée à une nouvelle entrée x , la méthode des k plus proches voisins consiste à prendre en compte (de façon identique) les k échantillons d'apprentissage dont l'entrée est la plus proche de la nouvelle entrée x , selon une distance à définir. Puisque cet algorithme est basé sur la distance, la normalisation peut améliorer sa précision.

- Objectif du TP:

Dans ce TP on va implémenter K-NN sur un vrai jeu de données pour faire une classification multi-classes.

- Pour suivre ce TP il faut disposer de la bibliothèque **Scikit-Learn**.

On utilisera le célèbre jeu de données **MNIST**.

Scikit-Learn vient avec un ensemble de jeux de données prêt à l'emploi pour des fins d'expérimentations. Ces dataset sont regroupés dans le package `sklearn.datasets`

Notre classificateur et dans : « **`sklearn.neighbors`** »

- Data set utiliser :

MNIST est une base de données étiquetée propice pour un apprentissage supervisé. Dans l'image ci-dessus, pour chaque chiffre, on a sa représentation sous forme d'image ainsi que son étiquette. Par exemple, pour le dernier chiffre en bas à droite, l'étiquette vaut 9 vu qu'il s'agit du chiffre 9. La représentation de ces chiffres est normalisée à travers tout le jeu de données MNIST. Ainsi, chaque chiffre est codé dans un format 8 pixels * 8 pixels. En plus, chaque pixel peut prendre une valeur de 0 à 255. Cette plage de valeurs représente le niveau de gris Grayscale. En d'autres termes, chaque représentation **d'une image est une matrice** de dimension. Le jeu de données MNIST présent par défaut dans la librairie Scikit Learn, comporte un sous-ensemble de la "vraie" base de données MNIST. Le sous-ensemble comporte chiffres que nous diviserons par la suite en deux sous ensembles : d'entraînement et de *test*.

1. Importation de librairies :

```
from sklearn.datasets import * # chargement du package datasets contenant plusieurs jeu de données
import pandas as pd # Chargement de Pandas
import matplotlib.pyplot as plt # import de Matplotlib
import numpy as np
from sklearn.model_selection import train_test_split # classe utilitaire pour découper les jeux de données
from sklearn.neighbors import KNeighborsClassifier # import de la classe de K-NN
# l'interface graphic
from PyQt5.QtWidgets import (QMainWindow,QComboBox, QTextEdit, QVBoxLayout, QPushButton, QAction, QFileDialog, QApplication)
from PyQt5.QtGui import QIcon
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PIL.ImageQt import ImageQt
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
from matplotlib.figure import Figure
from PIL import Image
```

2. Les Fonction utiliser dans le prétraitement :

```
def apprentissage(self):
    KNN.fit(x_train, y_train)
    print("apprentissage a ete bien fait")
```

```
#Méthode displayImage pour afficher des données images (méthode optionnelle)
def displayImage(self,i):
    plt.imshow(digit['images'][i], cmap='Greys_r')
    plt.show()
```

```
#####input##### pour inserer le id de l'image a tester
def inputdialogue(self):

    self.roll, self.done = QtWidgets.QInputDialog.getInt(
        self.centralwidget, 'Image de test', 'choisissez de 1347:1797')
    fig = Figure(figsize=(5, 4), dpi=100)
    self.displayImage(self.roll)
    sc=plt.imshow(digit['images'][self.roll], cmap='Greys_r')
    imagel = Image.fromarray(np.uint8( sc.get_cmap()(sc.get_array())*255))
    if (self.done):
        qimage = ImageQt(imagel)
        pixmap = QtGui.QPixmap.fromImage(qimage)
        pixmap5 = pixmap.scaled(100, 100,QtCore.Qt.KeepAspectRatio)
```

```
def test(self):
    #la précision par rapport aux données de test
    print(KNN.score(x_test,y_test))
    #Afficher un élément de la matrice format image
    test = np.array(digit['data'][self.roll])
    test1 = test.reshape(1,-1)
    #prédiction
    k=KNN.predict(test1)
    print(k)
    self.label_7.setText("l'image que vous avez choisi est correspond au nombre : "+str(k))
```

3. La Fonction Main:

```
if __name__ == "__main__":
    import sys

    digit = load_digits() # chargement du dataset MNIST
    dig = pd.DataFrame(digit['data'][0:1797]) # Création d'un dataframe Panda
    dig.head() # affiche le tableau ci-dessous
    plt.imshow(digit['images'][0], cmap='Greys_r')
    plt.show() # affichage de la première image du jeu de données MNIST
    digit.keys()
    train_x = digit.data # les input variables
    train_y = digit.target # les étiquettes (output variable)
    #découpage du jeu de données : 75% en Training set & 25% en Testing set
    x_train,x_test,y_train,y_test=train_test_split(train_x,train_y,test_size=0.25,shuffle=False)
    KNN = KNeighborsClassifier(7)# on veut entrainer un 7-NN Classifieur (on utilise 7 voisins)
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

4. Le Test :

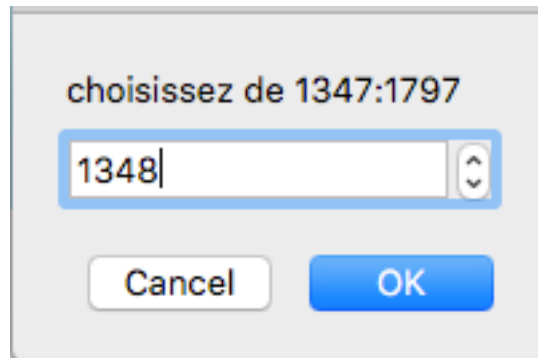
Après l'exécution on trouve l'interface suivante :



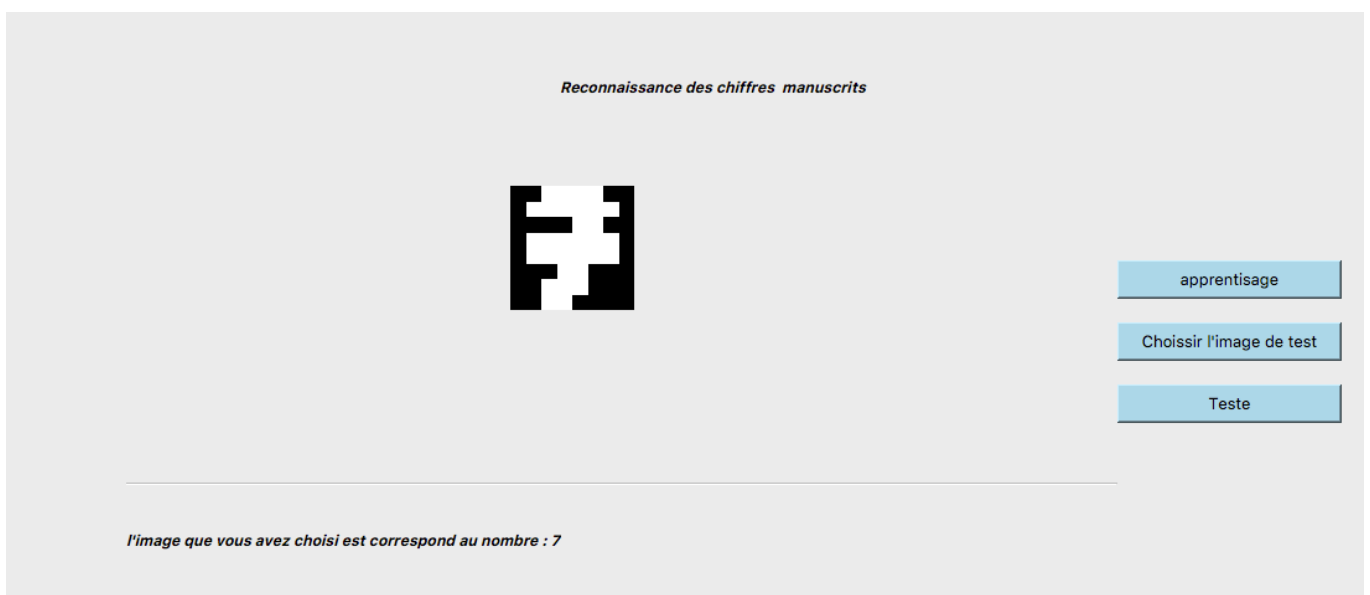
On clique sur apprentissage on obtient le message suivant dans le Console :

`apprentissage a ete bien fait`

On clique sur « choisir l'image de test » on obtient :



On constate que l'App a arrivé de reconnaitre le chiffre 7 :



Pourcentage de reconnaissance :

`0.9555555555555556`