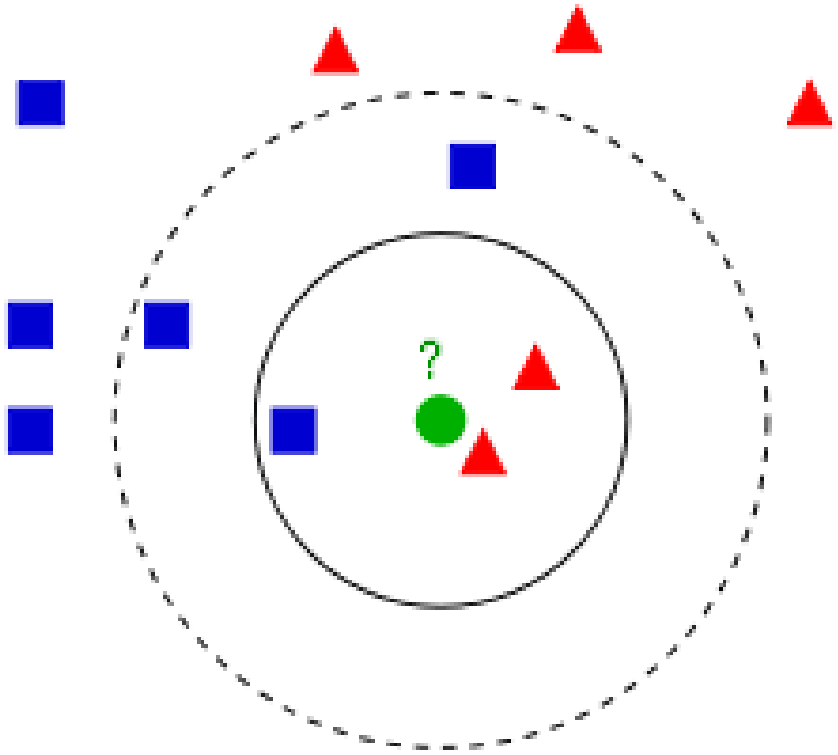
Promotion 2023

Année A3

Projet DataScience & IA année 3

Souhail AIT LAHCEN

Rapport concernant le projet KNN de DataScience & IA
Année 3



04/2021 - Document confidentiel

Table des matières

I Présentation du Projet 1
II Les méthodes de **Lecture des fichiers** 1
III Partie Commune 2
IV Partie 1 : Étude et construction du **TrainSet** et du **TestSet** avec seulement le fichier Data 3
V Partie 2 : Étude et construction du **TrainSet** et du **TestSet** avec les fichiers Data et preTest 6
VI Partie 3 : Étude et construction du **TrainSet** et du **TestSet** avec les fichiers Data et finalTest 9
VII Les Comparaisons avec d'autres fichiers 12
VIII Conclusion et Critiques 13

Rapport du Projet KNN de DataScience & IA Année 3
Année 2020-2021

Souhail Ait Lahcen

19 Avril 2021

I Présentation du Projet

Dans le cadre du deuxième problème de programmation de DataScience & IA nous devions réaliser un algorithme **KNN** pour déterminer les K-voisins plus proches d'un individu. L'objectif est donc de déterminer la classe et donc l'appartenance à une classe d'un individu en fonction de ses k-voisins. Il fallait donc créer les méthodes et/ou classes associées à ce projet.
Ainsi, l'objectif de ce rapport est de présenter les différentes méthodes et classes, que j'ai créé en lien avec le projet mais aussi faire part des problèmes que j'ai rencontrés.

Dans chaque sections et/ou sous-sections, nous pourrons utiliser des noms d'attributs associés aux classes appelées, ainsi veuillez les retrouver en annexe ou bien dans le programme Python.

Pour terminer cette introduction, vous devez savoir au préalable que vous pouvez démarrer le projet sur **Google collab** en suivant ce lien :
`https://colab.research.google.com/drive/1N6bweu3eymOvJHbqfcnyh4F91ferglkS?usp=sharing`.
Dans ce **Google collab**, les 3 parties sont référencés alors que dans l'algo **.py** nous avons seulement la partie 3 et les études pour obtenir le meilleur k possibles, ainsi certains Algo présentés dans les parties suivantes ne sont apparentes que sur le **Google collab** et non sur le fichier **.py** à la demande de l'énoncé du sujet.

II Les méthodes de Lecture des fichiers

Avant de commencer, nous devons d'abord réfléchir à comment lire les fichiers **.csv** donner en paramètres en sachant que contrairement aux fichiers données pour l'algorithme génétique, ici les fichiers n'ont pas de noms de colonnes et ainsi si on essaie de les lire sans nommer les colonnes alors la première ligne deviendra par défaut le nom des 7 colonnes.

Bien sûr, au début, je lisais qu'un seul fichier, par la suite j'ai crée trois versions de cette méthode, l'une qui ne lit que le fichier **Data**, l'autre qui lit les fichiers **Data** et **preTest** et le dernier qui lit les fichiers **Data**,**preTest** et **finalTest** comme vous pourrez le voir dans les parties suivantes.

Cette méthode renvoie donc les DataFrame correspondants aux fichiers. Cependant, comme vous aurez pu le remarquer la dernière version de cette méthode pour le fichier **finalTest** on a que 6 noms de colonnes car la dernière colonne qui correspond à la colonne de la classe n'apparaît pas dans le fichier **finalTest**.

Aussi pour information, les méthodes sont associés au package **pandas** disponibles sur **Google Collab** ou **Spyder**, qui permet de lire un fichier CSV.

III Partie Commune

Dans la partie commune, nous avons les librairies utilisés mais aussi les méthodes par rapport aux distances arithmétique des points. Nous avons utilisé, deux types de distances qui sont :

```
import time
import random
import numpy as np
from math import *
import csv
import pandas as pd
import matplotlib.pyplot as plt

# Partie Commune :


def DistancedeManhattan(X,Y,Z,U,V,W,x,y,z,u,v,w):
    a=abs(X-x)
    b=abs(Y-y)
    c=abs(Z-z)
    d=abs(U-u)
    e=abs(V-v)
    f=abs(W-w)
    g=a+b+c+d+e+f
    return g



def DistanceEclidienne(X,Y,Z,U,V,W,x,y,z,u,v,w):
    a=(X-x)**2
    b=(Y-y)**2
    c=(Z-z)**2
    d=(U-u)**2
    e=(V-v)**2
    f=(W-w)**2
    g=sqrt(a+b+c+d+e+f)
    return g
```

Il y'a donc deux méthodes, celle qui calcule la distance de **Manhattan** entre deux points et celle qui calcule la distance **Euclidienne**. La distance **Euclidienne** permet de généraliser l'application du théorème de **Pythagore** à un espace de dimension n. C'est la distance la plus « intuitive ». Alors que la distance de **Manhattan** est indépendante du chemin parcouru à l'intérieur d'un réseau fini. Il existe aussi plusieurs types de distances.

Distance sur des espaces vectoriels [modifier | modifier le code]

Sur un espace vectoriel normé $(E, \|\cdot\|)$, la distance d « induite par » la norme[5] $\|\cdot\|$ est définie par :

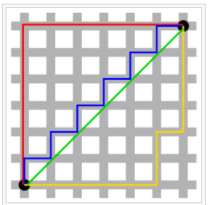
$$\forall (x, y) \in E \times E \quad d(x, y) = \|y - x\|.$$

En particulier, dans $\mathbb{R}^n$, on peut définir de plusieurs manières la distance entre deux points, bien qu'elle soit généralement donnée par la distance euclidienne (ou *2-distance*). Soit deux points de E, $(x_1, x_2, \ldots, x_n)$ et $(y_1, y_2, \ldots, y_n)$, on exprime les différentes distances ainsi :

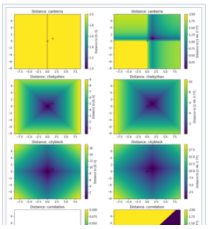
Nom	Paramètre	Fonction
distance de Manhattan	1-distance	$\sum_{i=1}^{n} \|x_i - y_i\|$
distance euclidienne	2-distance	$\sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$
distance de Minkowski	p-distance	$\sqrt[p]{\sum_{i=1}^{n} \|x_i - y_i\|^p}$
distance de Tchebychev	∞-distance	$\lim_{p \to \infty} \sqrt[p]{\sum_{i=1}^{n} \|x_i - y_i\|^p} = \sup_{1 \leq i \leq n} \|x_i - y_i\|$

La 2-distance permet de généraliser l'application du théorème de Pythagore à un espace de dimension n. C'est la distance la plus « intuitive ».

La p-distance est rarement utilisée en dehors des cas p = 1, 2 ou ∞. L'∞-distance présente la particularité amusante de permettre la définition en toute rigueur de sphères **cubiques** (voir oxymore). La 1-distance permet de définir des sphères octaédriques.



Distance de Manhattan (chemins rouge, jaune et bleu) contre distance euclidienne en vert.



Cependant, comme le montre les deux images ci-dessous :

```
In [64]: Algo()
Le pourcentage de réussite est :  91.82389937106919
Temps : 5.8733296394348145 secondes

In [65]: runcell('Partie 1 : Etudie et construction du TrainSet et du Te
avec seulement le fichier Data', 'D:/ESILV/Projet IA/Projet KNN/Projet K
V2.py')

In [66]: Algo()
Le pourcentage de réussite est :  91.57303370786516
Temps : 6.049821853637695 secondes
```

La distance de **Manhattan** permet une meilleure précision (Pour la partie 1 et elle correspond au premier pourcentage) de l'algorithme comme le montre les résultats, et ceci pour les parties 1 et 2 ci-après.
Ainsi pour le reste des parties et notamment la dernière partie c'est celle-ci qui sera retenue (bien évidemment les algorithmes de chaque parties dispose d'une ligne vous permettant d'utiliser la distance **Euclidienne**).

IV Partie 1 : Étude et construction du TrainSet et du TestSet avec seulement le fichier Data

Comme précisé dans l'énoncé du projet, nous devons étudier à la fois un **TrainSet** de valeurs et un **TestSet** de valeurs avec les valeurs du fichier Data.csv donné en source sur **DVO**.

```
#%% Partie 1 : Etudie et construction du TrainSet et du TestSet avec seulement le fichier Data

def LireCSV():
    df=pd.read_csv("D:/ESILV/Projet IA/Projet KNN/data.csv",sep=',',names=['X','Y','Z','U','V','W','Classe'], header=None)
    return df

# On va diviser selon un nombre alétoire compris entre 15% et 25% le dataset
def Pourcentage():
    a=random.uniform(0,1)
    while a<0.75 or a>0.85:
        a=random.uniform(0,1)
    return a

 #On retourne le nouveau dataset qu'on va étudier
def PartitionEnListe(df,a):
    b=round(a*len(df))
    TrainSet=df.iloc[:b]
    TestSet=df.iloc[b:len(df)]
    return TrainSet,TestSet,b

def Comparer(df,LC,Pos,k):
    x=0
    compteur=0
    l=0
    for j in range(len(Pos)):
        for i in range(k):
            if(df["Classe"][LC[l][i]]==df["Classe"][Pos[j]]):
                x+=1
            if(x>=round(k/2)):
                compteur+=1
                break
        x=0
        l+=1
    pourcentage=((compteur/len(Pos))*100)
    return pourcentage
```

Tout d'abord, nous devons lire le fichier Data.csv et seulement travaillé dessus avec. La première fonction **LireCSV()**, permet de lire le fichier, la deuxième **Pourcentage()** permet d'obtenir un pourcentage de valeur contenu dans le **TrainSet** et le **TestSet**, le **TrainSet** va contenir plus au moins 80% des valeurs du fichier grâce à la troisième fonction **PartitionEnListe(df,a)**. Nous avons aussi une quatrième fonction mais celle-ci sera détaillée plus bas.
Ensuite, passons à la construction de l'algo qui sera, à quelques changements près, quasiment le même entre les trois parties.

Premièrement, de manière arbitraire, j'ai choisi un **K = 3**, pour comprendre et simplifier la structure de l'algorithme dans un premier temps. Cet algorithme, va "travailler" sur les deux Dataframe crées grâce à **PartitionEnListe(df,a)**.
Il compare chacune des valeurs du **TestSet** au **TrainSet** et sauvegarde les trois meilleurs positions pour chaque indice en fonction de leurs distances par rapport à l'individu étudié. Après, la sortie de ces deux boucles nous appelons la quatrième fonction précédemment citée qui est **Comparer(df,LC,Pos,k)** qui va vérifier si la majorité des postions donnée pour un individu on la même classe réellement. L'algorithme, nous dit alors à la fin le pourcentage de réussite. Voici l'algo en question :

```
def Algo():
    debut = time.time()
    df=LireCSV()
    a=Pourcentage()
    TrainSet,TestSet,b=PartitionEnListe(df,a)
    LC=[]
    Pos=[]
    beta=[]
    k = 3
    for i in range(b,len(df["X"])):
        beta = []
        for j in range(len(TrainSet["X"])):
            X=TrainSet["X"][j]
            Y=TrainSet["Y"][j]
            Z=TrainSet["Z"][j]
            U=TrainSet["U"][j]
            V=TrainSet["V"][j]
            W=TrainSet["W"][j]
            x=TestSet["X"][i]
            y=TestSet["Y"][i]
            z=TestSet["Z"][i]
            u=TestSet["U"][i]
            v=TestSet["V"][i]
            w=TestSet["W"][i]
            beta.append([DistancedeManhattan(X,Y,Z,U,V,W,x,y,z,u,v,w), i, j])
            #beta.append([DistanceEclidienne(X,Y,Z,U,V,W,x,y,z,u,v,w), i, j])
        beta = sorted(beta, key = lambda x : x[0])
        best=[beta[x][-1] for x in range(k)]
        #print("le meilleur triplet pos newdf",best," la valeur de pos de newdf2 associé :",beta[b-i][1])
        LC.append(best)
        Pos.append(beta[b-i][1])
    pourcentage=Comparer(df, LC,Pos,k)
    print("Le pourcentage de réussite est : ",pourcentage)
    fin = time.time()
    print("Temps :",fin-debut,"secondes")
```

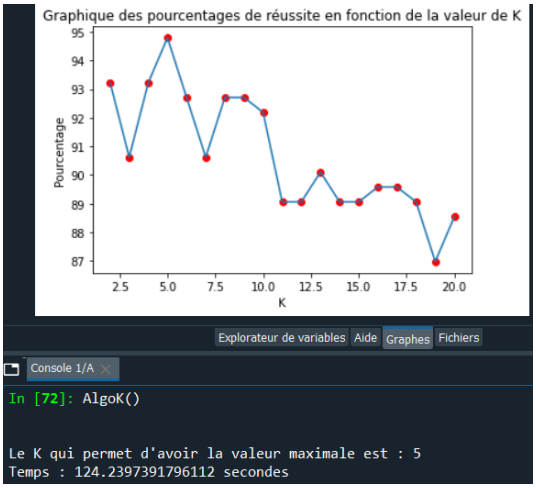
Deuxièmement, afin d'avoir les meilleurs voisins nous sommes obligés de créer un algo qui nous permet d'obtenir le meilleur **K** possible, pour cela nous allons reprendre le précédent algorithme mais en faisant varier notre **K** de 2 à 20, pour obtenir le meilleur nous allons réaliser un graphique qui va nous permettre d'observer quel est le meilleur **K**. Le meilleur **K** sera celui qui aura le plus haut taux de réussite. Voici, les deux fonctions qui nous permettent cette observation :

```
def AlgoK():
    debut = time.time()
    df=LireCSV()
    a=Pourcentage()
    TrainSet,TestSet,b=PartitionEnListe(df,a)
    LC=[]
    Pos=[]
    beta=[]
    LesK=[]
    LesPourcentages=[]
    for k in range(2,21):
        LC=[]
        Pos=[]
        for i in range(b,len(df["X"])):
            beta = []
            for j in range(len(TrainSet["X"])):
                X=TrainSet["X"][j]
                Y=TrainSet["Y"][j]
                Z=TrainSet["Z"][j]
                U=TrainSet["U"][j]
                V=TrainSet["V"][j]
                W=TrainSet["W"][j]
                x=TestSet["X"][i]
                y=TestSet["Y"][i]
                z=TestSet["Z"][i]
                u=TestSet["U"][i]
                v=TestSet["V"][i]
                w=TestSet["W"][i]
                beta.append([DistancedeManhattan(X,Y,Z,U,V,W,x,y,z,u,v,w), i, j])
                #beta.append([DistanceEclidienne(X,Y,Z,U,V,W,x,y,z,u,v,w), i, j])
            beta = sorted(beta, key = lambda x : x[0])
            best=[beta[x][-1] for x in range(k)]
            #print("le meilleur triplet pos newdf",best," la valeur de pos de newdf2 associé :",beta[b-i][1])
            LC.append(best)
            Pos.append(beta[b-i][1])
        pourcentage=Comparer(df, LC,Pos,k)
        print("Le pourcentage de réussite est : ",pourcentage,"et le k correspondant est :",k)
        LesK.append(k)
        LesPourcentages.append(pourcentage)
    print("\n")
    print("Le K qui permet d'avoir la valeur maximale est :",LesK[LesPourcentages.index(max(LesPourcentages))])
```

```
    print("\n")
    print("Le K qui permet d'avoir la valeur maximale est :",LesK[LesPourcentages.index(max(LesPourcentages))])
    Graphique(LesK,LesPourcentages)
    fin = time.time()
    print("Temps :",fin-debut,"secondes")

def Graphique(x,y):
    plt.title("Graphique des pourcentages de réussite en fonction de la valeur de K")
    plt.xlabel('K')
    plt.ylabel('Pourcentage')
    plt.plot(x,y)
    plt.scatter(x,y,c = 'red')
    plt.show()
```

Voici, ce que le graphique nous affiche :



Pour finir, le meilleur **K** pour cette partie est **K = 5** et ce sera aussi le cas dans la prochaine partie.
Nous allons donc passer à la partie de l'explication des méthodes qui nous permettent de faire la partie 2 : Étude et construction du **TrainSet** et du **TestSet** avec les fichiers Data et preTest.

V Partie 2 : Étude et construction du TrainSet et du TestSet avec les fichiers Data et preTest

Maintenant, nous devons étudier à la fois un **TrainSet** de valeurs et un **TestSet** de valeurs avec les valeurs des fichiers Data et preTest.

```
#%% Partie 2 : Etudie et construction du TrainSet et du TestSet avec les fichiers Data et preTest

def LireCSV_2():
    TrainSet=pd.read_csv("D:/ESILV/Projet IA/Projet KNN/data.csv",sep=',',names=['X','Y','Z','U','V','W','Classe'], header=None)
    TestSet=pd.read_csv("D:/ESILV/Projet IA/Projet KNN/preTest.csv",sep=',',names=['X','Y','Z','U','V','W','Classe'], header=None)
    return TrainSet,TestSet

def Comparer_2(TrainSet,TestSet,LC,Pos,k):
    x=0
    compteur=0
    l=0
    for j in range(len(Pos)):
        for i in range(k):
            if(TrainSet["Classe"][LC[l][i]]==TestSet["Classe"][Pos[j]]):
                x+=1
            if(x>=round(k/2)):
                compteur+=1
                break
        x=0
        l+=1
    pourcentage=((compteur/len(Pos))*100)
    return pourcentage
```

Tout d'abord, comme dans la précédente partie nous allons lire le fichier Data.csv mais aussi le fichier preTest.csv et travailler dessus. La première fonction **LireCSV_2()**, permet de lire le fichier, la deuxième sera détaillée plus bas.
Ensuite, passons à la construction de l'algo.
Premièrement, grâce à la comparaison dans l'algo suivant mais aussi dans la partie précédente, j'ai choisi un **K = 5**. Cet algorithme, va "travailler" sur les deux Dataframe. Il compare chacune des valeurs du **TestSet** au **TrainSet** et sauvegarde les **K** meilleurs positions pour chaque indice en fonction de leurs distances par rapport à l'individu étudié.
Après, la sortie de ces deux boucles nous appelons la quatrième fonction précédemment citée qui est **Comparer_2(TrainSet,TestSet,LC,Pos,k)** qui va vérifier si la majorité des positions donnée pour un individu on l'a même classe réellement.
Ce qui signifie si par exemple on a une liste l=["ClassA","ClassB","ClassA","ClassA","ClassC"] pour un individu donnée et que cet individu est un classe A.
Alors, si la majorité soit dans notre cas, qu'il y'ait 3 éléments ou plus qui est une classeA alors on a bon sinon on revoit soit une classe une hasard soit une classe qui est fausse.
L'algorithme, nous dit alors à la fin le pourcentage de réussite. Voici l'algo en question :

```
def Algo2():
    debut = time.time()
    TrainSet,TestSet=LireCSV_2()
    LC=[]
    Pos=[]
    beta=[]
    k = 5
    for i in range(len(TestSet["X"])):
        beta = []
        for j in range(len(TrainSet["X"])):
            X=TrainSet["X"][j]
            Y=TrainSet["Y"][j]
            Z=TrainSet["Z"][j]
            U=TrainSet["U"][j]
            V=TrainSet["V"][j]
            W=TrainSet["W"][j]
            x=TestSet["X"][i]
            y=TestSet["Y"][i]
            z=TestSet["Z"][i]
            u=TestSet["U"][i]
            v=TestSet["V"][i]
            w=TestSet["W"][i]
            beta.append([DistancedeManhattan(X,Y,Z,U,V,W,x,y,z,u,v,w), i, j])
            #beta.append([DistanceEclidienne(X,Y,Z,U,V,W,x,y,z,u,v,w), i, j])
        beta = sorted(beta, key = lambda x : x[0])
        best=[beta[x][-1] for x in range(k)]
        #print("le meilleur triplet pos newdf",best," la valeur de pos de newdf2 associé :",beta[i][1])
        LC.append(best)
        Pos.append(beta[i][1])
    pourcentage=Comparer_2(TrainSet,TestSet, LC,Pos,k)
    print("Le pourcentage de réussite est : ",pourcentage)
    fin = time.time()
    print("Temps :",fin-debut,"secondes")
```

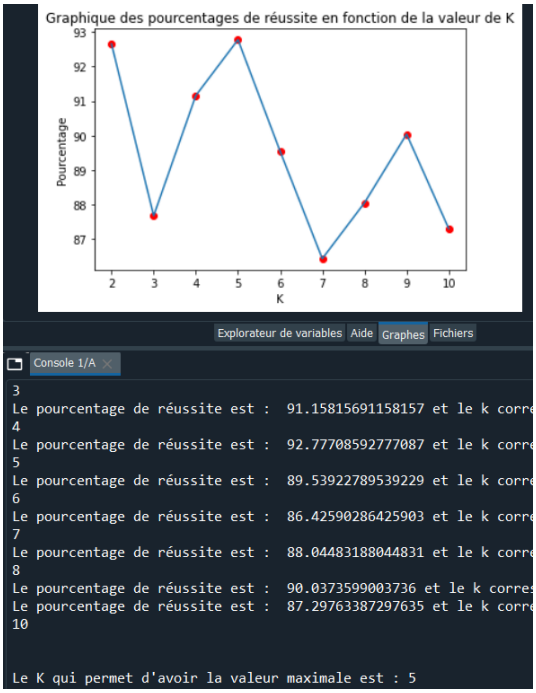
Deuxièmement, afin d'avoir les meilleurs voisins nous sommes obligés de créer un algo qui nous permet d'obtenir le meilleur **K** possible, pour cela nous allons reprendre le précédent algorithme qui fait varier **K** mais avec les fichiers Data et preTest en faisant varier notre **K** de 2 à 20, pour obtenir le meilleur nous allons réaliser un graphique qui va nous permettre d'observer quel est le meilleur **K**. Le meilleur **K** sera celui qui aura le plus haut taux de réussite. Voici, les deux fonctions qui nous permettent cette observation :

```
def AlgoK2():
    debut = time.time()
    TrainSet,TestSet=LireCSV_2()
    LC=[]
    Pos=[]
    beta=[]
    LesK=[]
    LesPourcentages=[]
    for k in range(2,21):
        LC=[]
        Pos=[]
        for i in range(len(TestSet["X"])):
            beta = []
            for j in range(len(TrainSet["X"])):
                X=TrainSet["X"][j]
                Y=TrainSet["Y"][j]
                Z=TrainSet["Z"][j]
                U=TrainSet["U"][j]
                V=TrainSet["V"][j]
                W=TrainSet["W"][j]
                x=TestSet["X"][i]
                y=TestSet["Y"][i]
                z=TestSet["Z"][i]
                u=TestSet["U"][i]
                v=TestSet["V"][i]
                w=TestSet["W"][i]
                beta.append([DistancedeManhattan(X,Y,Z,U,V,W,x,y,z,u,v,w), i, j])
                #beta.append([DistanceEclidienne(X,Y,Z,U,V,W,x,y,z,u,v,w), i, j])
            beta = sorted(beta, key = lambda x : x[0])
            best=[beta[x][-1] for x in range(k)]
            #print("le meilleur triplet pos newdf",best," la valeur de pos de newdf2 associé :",beta[i][1])
            LC.append(best)
            Pos.append(beta[i][1])
        pourcentage=Comparer_2(TrainSet,TestSet, LC,Pos,k)
        print("Le pourcentage de réussite est : ",pourcentage,"et le k correspondant est :",k)
        LesK.append(k)
        LesPourcentages.append(pourcentage)
    print("\n")
    print("Le K qui permet d'avoir la valeur maximale est :",LesK[LesPourcentages.index(max(LesPourcentages))])
    Graphique(LesK,LesPourcentages)
    fin = time.time()
```

```
    Graphique(LesK,LesPourcentages)
    fin = time.time()
    print("Temps :",fin-debut,"secondes")


def Graphique2(x,y):
    plt.title("Graphique des pourcentages de réussite en fonction de la valeur de K")
    plt.xlabel('K')
    plt.ylabel('Pourcentage')
    plt.plot(x,y)
    plt.scatter(x,y,c = 'red')
    plt.show()
```

Voici, ce que le graphique nous affiche :



Pour finir, le meilleur **K** pour cette partie est **K = 5** comme dans la précédente partie. Nous allons donc passer à la Partie 3 : Étude et construction du **TrainSet** et du **TestSet** avec les fichiers Data et finalTest.

VI Partie 3 : Étude et construction du TrainSet et du TestSet avec les fichiers Data et finalTest

Dans cette partie, nous allons expliquer l'algorithme finale, en donnée nous allons prendre les fichiers Data, preTest et finalTest et grâce à la fonction **merge** qui va nous permettre d'avoir en un dataframe les données de **Data** et **preTest** afin d'avoir la plus grande base de données. On retrouve toujours les mêmes fonctions que dans les précédentes parties donc **LireCSV_3()** et **Comparer_3(TrainSet,LC,Pos,k)**.

```python
#%% Partie 3 (partie final) : Etudie et construction du TrainSet et du TestSet avec les fichiers Data,preTest et finalTest

def LireCSV_3():
    TrainSet1=pd.read_csv("D:/ESILV/Projet IA/Projet KNN/data.csv",sep=',',names=['X','Y','Z','U','V','W','Classe'], header=None)
    TrainSet2=pd.read_csv("D:/ESILV/Projet IA/Projet KNN/preTest.csv",sep=',',names=['X','Y','Z','U','V','W','Classe'], header=None)
    TrainSet=pd.merge(TrainSet1,TrainSet2,how = 'outer')
    TestSet=pd.read_csv("D:/ESILV/Projet IA/Projet KNN/finalTest.csv",sep=',',names=['X','Y','Z','U','V','W'], header=None)
    return TrainSet,TestSet

def Comparer_3(TrainSet,LC,Pos,k):
    x=1
    compteur=0
    l=0
    Classe=[]
    for j in range(len(Pos)):
        for i in range(k):
            if(i<k-1):
                if(TrainSet["Classe"][LC[l][i]]==TrainSet["Classe"][LC[l][i+1]]):
                    x+=i
            else:
                if(TrainSet["Classe"][LC[l][i-1]]==TrainSet["Classe"][LC[l][i]]):
                    x+=i
                if(x>=round(k/2)):
                    compteur+=1
                    Classe.append(TrainSet["Classe"][LC[l][i-1]])
                    break
                if(x<round(k/2)):
                    index=random.randint(0,k-1)
                    Classe.append(TrainSet["Classe"][LC[l][index]])
                    break
        x=0
        l+=1
    pourcentage=((compteur/len(Pos))*100)
    return pourcentage,Classe
```

Ensuite, passons à la construction de l'algo.

Premièrement, grâce à la comparaison des deux précédentes parties, j'ai donc choisi **K = 5**. Cet algorithme, va "travailler" sur les trois Dataframe. Il compare chacune des valeurs du **TestSet** au **TrainSet** et sauvegarde les **5** meilleurs positions (les plus proches) pour chaque indice en fonction de leurs distances par rapport à l'individu étudié.
Après, la sortie de ces deux boucles nous appelons la quatrième fonction précédemment citée qui est **Comparer_3(TrainSet,LC,Pos,k)** qui va vérifier si la majorité des positions donnée pour un individu on l'a même classe.
Ce qui signifie si par exemple on a une liste l=["ClassA","ClassB","ClassA","ClassA","ClassC"] pour un individu donnée alors en majorité pour l'algo l'individu appartient à la ClassA si l'algo ne peut pas déterminer par majorité alors il attribue au hasard à l'individu une classe présente dans la liste.

L'algorithme, nous dit alors à la fin un pourcentage d'estimation de réussite basé sur la majorité ou pas obtenu pour chaque individu étudié. Voici l'algo en question :

```
def Algo3():
    debut = time.time()
    TrainSet,TestSet=LireCSV_4()
    LC=[]
    Pos=[]
    beta=[]
    k = 5
    compteur=0
    for i in range(len(TestSet["X"])):
        beta = []
        for j in range(len(TrainSet["X"])):
            X=TrainSet["X"][j]
            Y=TrainSet["Y"][j]
            Z=TrainSet["Z"][j]
            U=TrainSet["U"][j]
            V=TrainSet["V"][j]
            W=TrainSet["W"][j]
            x=TestSet["X"][i]
            y=TestSet["Y"][i]
            z=TestSet["Z"][i]
            u=TestSet["U"][i]
            v=TestSet["V"][i]
            w=TestSet["W"][i]
            beta.append([DistancedeManhattan(X,Y,Z,U,V,W,x,y,z,u,v,w), i, j])
            #beta.append([DistanceEclidienne(X,Y,Z,U,V,W,x,y,z,u,v,w), i, j])
        compteur+=1
        if(compteur==len(TrainSet["X"])):
            compteur=0
        if(i<len(TrainSet["X"])):
            beta = sorted(beta, key = lambda x : x[0])
            best=[beta[x][-1] for x in range(k)]
            #print("Les plus proches voisins selon k =",k,"sont :",best,"et l'index de l'élément dans le TestSet associé est :",beta[i][1])
            LC.append(best)
            Pos.append(beta[i][1])
        else:
            beta = sorted(beta, key = lambda x : x[0])
            best=[beta[x][-1] for x in range(k)]
            #print("Les plus proches voisins selon k =",k,"sont :",best,"et l'index de l'élément dans le TestSet associé est :",beta[compteur][1])
            LC.append(best)
```

```
        else:
            beta = sorted(beta, key = lambda x : x[0])
            best=[beta[x][-1] for x in range(k)]
            #print("Les plus proches voisins selon k =",k,"sont :",best,"et l'index de l'élément dans le TestSet associé est :",beta[compteur][1])
            LC.append(best)
            Pos.append(beta[compteur][1])
    pourcentage,Classe=Comparer_4(TrainSet, LC,Pos,k)
    print("Le pourcentage d'estimation de réussite, qui se base la majorité de la classe des plus proches voisins selon le k, est : ",pourcentage,"%.
    EcrireFichier2(Classe)
    fin = time.time()
    print("Temps :",fin-debut,"secondes")



def EcrireFichier3(x):
    fichier=open("D:/ESILV/Projet IA/Projet KNN/FichierdedataFK52.txt","w")
    for i in range(len(x)):
        if(i<len(x)-1):
            fichier.write(str(x[i]))
            fichier.write("\n")
        else:
            fichier.write(str(x[i]))
    fichier.close()
```

Enfin, nous pouvons écrire dans un fichier les classes obtenues dans un fichier grâce à la fonction **EcrireFichier3(x)** qui prend en paramètre la liste des classes et l'écrit sous format **.txt** comme vous pouvez le voir ci-dessus.

Aussi, dans la cadre du projet nous devions vérifier grâce à la fonction donnée vérifier que notre fichier texte respecte les attendus et comme vous pouvez le voir ci-dessus le fichier respect bien les attentes.
Voici, la fonction :

```
import sys

#code permettant de tester si un fichier de prédictions est au bon format.
#il prend en paramètre un fichier de labels prédits
#exemple> python checkLabels.py mesPredictions.txt

Souhail="D:/ESILV/Projet IA/Projet KNN/FichierdedataFK52.txt"

allLabels = ['classA','classB','classC','classD','classE']
#ce fichier s'attend à lire 3000 prédictions, une par ligne
#réduisez nbLines en période de test.
nbLines = 3000
fd =open(Souhail,'r')
lines = fd.readlines()


count=0
for label in lines:
    if label.strip() in allLabels:
        count+=1
    else:
        if count<nbLines:
            print("Wrong label line:"+str(count+1))
            break
if count<nbLines:
    print("Labels Check : fail!")
else:
    print("Labels Check : Successfull!")
```

Et le résultat associé :

```
In [91]: runcell(0, 'D:/ESILV/Projet IA/Projet KNN/checkLabels.py')
Labels Check : Successfull!
```

Nous allons donc passer à la partie Les Comparaisons avec d'autres fichiers.

VII Les Comparaisons avec d'autres fichiers

Dans un cadre de recherche, j'ai utilisé une méthode permettant de comparer entre deux fichiers afin de pouvoir obtenir le pourcentage de confusions et voir si les valeurs sont cohérentes en sachant que c'est notre seul moyen de savoir si nous avons bons ou pas car nous ne connaissons pas les résultats réels. Voici la fonction :

```
points_louis = []
points_sousou = []
cpt = 0
percentage = 0.0

Souhail,S="D:/ESILV/Projet IA/Projet KNN/FichierdedataF.txt","Souhail"
Souhail2,S2="D:/ESILV/Projet IA/Projet KNN/FichierdedataF22.txt","Souhail2"
Souhail3,S3="D:/ESILV/Projet IA/Projet KNN/FichierdedataFK5.txt","Souhail3"
Souhail4,S4="D:/ESILV/Projet IA/Projet KNN/FichierdedataFK52.txt","Souhail4"

Louis,L="D:/ESILV/Projet IA/Projet KNN/resultLouis.txt","louis"
Jean,J="D:/ESILV/Projet IA/Projet KNN/LeChevalier_jean.txt","Jean"
Jean2,J2="D:/ESILV/Projet IA/Projet KNN/LeChevalier_jean2.txt","Jean2"
Pierre,P="D:/ESILV/Projet IA/Projet KNN/FichierdePierre.txt","Pierre"
Nas,N="D:/ESILV/Projet IA/Projet KNN/Nasreddine_GRIHMA.txt","Nas"


with open(Nas, "r") as f:
    for l in f:
        points_louis.append(str(l))

with open(Souhail4, "r") as f:
    for l in f:
        points_sousou.append(str(l))

for k in range(len(points_louis)):
    if points_louis[k] == points_sousou[k]:
        cpt += 1

percentage = round((cpt/len(points_louis)) * 100, 2)
print("Pourcentage de confusion : %f" % percentage)
print("Entre les fichiers",N,"et",S)
```

et quelques valeurs de pourcentage de confusion entre mes camarades :

```
In [85]: runcell(0, 'D:/ESILV/Projet IA/Projet KNN/TestDesFichiers.py')
Pourcentage de confusion : 91.430000
Entre les fichiers Jean et Souhail

In [86]: runcell(0, 'D:/ESILV/Projet IA/Projet KNN/TestDesFichiers.py')
Pourcentage de confusion : 92.170000
Entre les fichiers louis et Souhail

In [87]: runcell(0, 'D:/ESILV/Projet IA/Projet KNN/TestDesFichiers.py')
Pourcentage de confusion : 91.300000
Entre les fichiers Pierre et Souhail

In [88]: runcell(0, 'D:/ESILV/Projet IA/Projet KNN/TestDesFichiers.py')
Pourcentage de confusion : 92.900000
Entre les fichiers Nas et Souhail
```

Nous allons donc passer à la conclusion.

VIII Conclusion et Critiques

Pour conclure, il y'a eu de nombreux problèmes dans l'écriture et tests des méthodes, entre les **sorties d'index**, l'efficacité des méthodes ou la lenteur de l'algorithme. Aussi, malheureusement, lors de l'écriture de ce projet, j'avais aussi perdu une partie de mon

projet à cause de l'ouverture du fichier **checkLabels** notamment qui s'est ouvert sur le fichier **.py** du projet. J'ai donc dû réécrire une partie de mon projet. Les problèmes étaient nombreux mais pas sans solutions, le projet a pu être construit correctement.