IA - Classification Challenge

1) Explication du code

```
def LoadDataSet(split, trainingSet, testSet):
    with open('data.csv','r') as csvfile:
        lines=csv.reader(csvfile)
        dataset=list(lines)
        #print(dataset) #matrice 803x7
        print(len(dataset)) #=803
        for x in range (len(dataset)):
            for y in range(6):
                dataset [x][y]=float(dataset[x][y])
            if random.random() < split:
                trainingSet.append(dataset[x])
            else:
                testSet.append(dataset[x])
        print(len(testSet)+len(trainingSet))
```

Nous avons à notre disposition un fichier data avec les noms individus associés aux caractéristiques de la fleurs. J'ai chargé les données avec la fonction LoadDataSet(), les données sont coupées en trainSet/testSet afin de l'entraîner, faire varier des paramètres de votre approche et de l'évaluer par la suite.

```
def Distance(instance1, instance2, length): #distance euclidienne
        dist=0
        for i in range(length):
            calcul= instance1[i] - instance2[i]
            dist= dist + pow(calcul, 2)
        return math.sqrt(dist)
```

La fonction Distance() va nous renvoyer la distance euclienne entre deux individus.

```
def Knn(trainingSet, testInstance, k):
    list_distances=[]
    length=len(testInstance)-1
    for i in range(len(trainingSet)):
        dist=Distance(testInstance, trainingSet[i], length)
        list_distances.append((trainingSet[i],dist))
    list_distances.sort(key=operator.itemgetter(1))
    list_neighbors=[]
    for i in range(k):
        list_neighbors.append(list_distances[i][0])
    return list_neighbors
```

Mon Knn() va renvoyer un tableau des k individus les plus ressemblants en se basant sur un critère de similitude qui est la distance eucliendienne.

```
def getResponse(list_neighbors):
    classVotes={}
    for i in range(len(list_neighbors)):
        resp= list_neighbors[i][-1]
        if resp in classVotes:
            classVotes[resp]+=1
        else:
            classVotes[resp] = 1
    sortedVotes= sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)
    #print(classVotes)
    return sortedVotes[0][0]
```

La fonction getResponse va nous permettre de faire notre prédictions ceci avec le tableau des k individus les plus ressemblants. A l'aide d'un dictionnaire nous allons identifier quel est l'individu le plus représenté parmi les individus similaires, ce sera notre prédiction.

```
def Accuracy(testSet, predictions): #précision des prédictions
    correct=0
    for i in range(len(testSet)):
        #if testSet[i][-1] is predictions[i]:
        if testSet[i][-1]== predictions[i]:
            correct= correct + 1
    return(correct/float(len(testSet)))*100.0
```

Accuracy() va être notre indicateur de performance de notre algorithme en donnant le % de prédictions exacts. Nous voulons tendre la précision des prédictions de notre algorithme vers 100%.

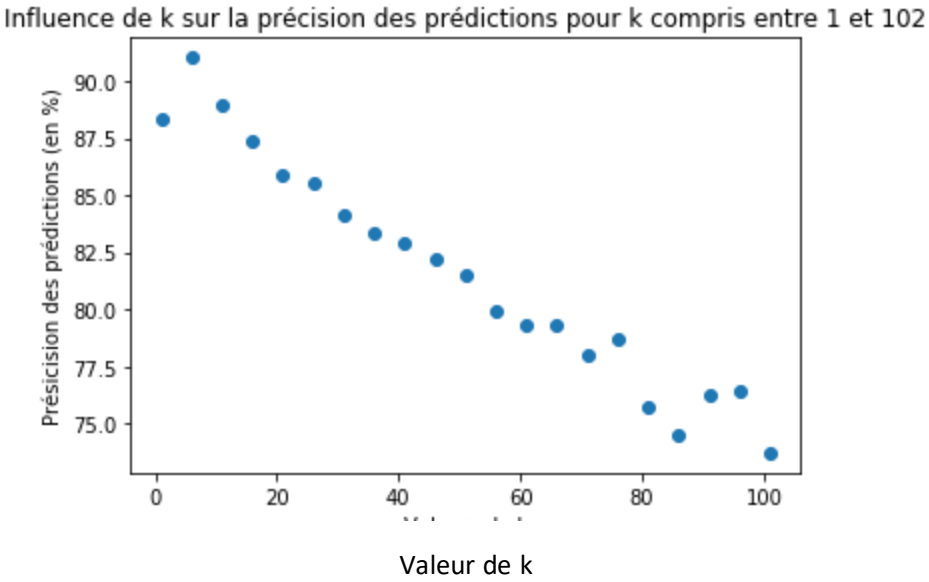
```
def Main(k):
    # prepare data
    trainingSet=[]
    testSet=[]
    split=0.90
    LoadDataSet(split, trainingSet, testSet)
    #print('Training: ' + str(len(trainingSet)))
    #print('Test: ' + str(len(testSet)))
    # generate predictions
    predictions=[]
    for x in range(len(testSet)):
        list_neighbors=Knn(trainingSet, testSet[x], k)
        resultat= getResponse(list_neighbors)
        predictions.append(resultat)
    accuracy= Accuracy(testSet, predictions)
    print('Accuracy: ' + str(accuracy) + '%')
    return accuracy
```

J'ai choisi de split à 0.90 pour préviliégier un plus grand nombre d'invidus dans le trainingSet. Il me semble pertinent de vouloir beaucoup d'individus, de données sur lesquelles l'algorithme pourra chercher pour avoir une meilleur précision

.

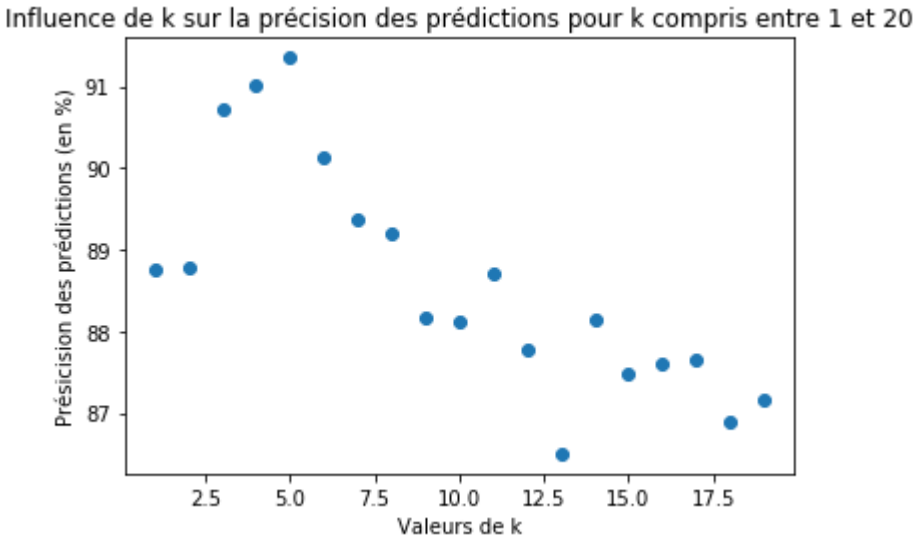
2) Recherche de k

J'ai voulu trouvé le k le plus performant pour notre algorithme knn grâce à la data donnée avec la classe spécifiée. J'ai implenté une fonction influencek() qui fait tourner l'algorithme knn 50 fois pour chaque k et en retourne la moyenne de ces 50 tours. La fonction graphek() a permit d'effectuer les graphes ci-dessous. J'ai donc fais varier k de 1 à 102 avec des pas de 5.



Nous voyons une tendance, la fiabilité des prédictions diminue au fur et à mesure que k augmente.

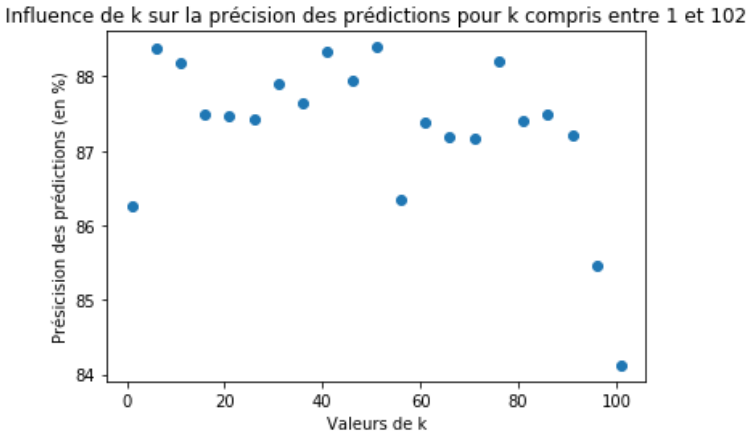
Le % de précision a varié de 91% à 74% avec un pique entre k=1 et k=20. J'ai donc par la suite tracé le graphe de l'inffluence de k entre 1 et 20 avec un pas de 1.



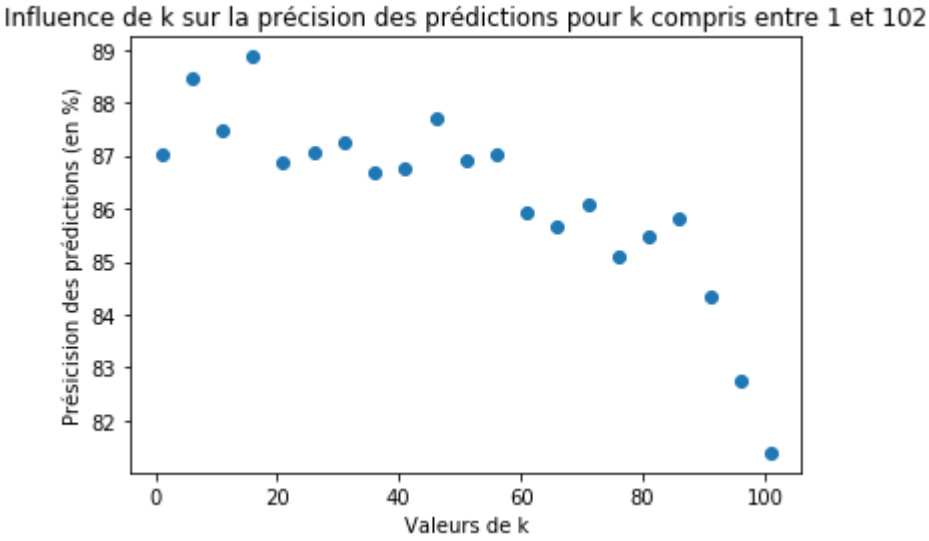
Nous retrouvons la tendance du graphe précèdent ce qui est rassurant. Nous avons un sommet en k=5. Nous pouvons extraire de ces graphes que k=5 est le plus performant pour nos prédictions.

3) Amélioration

Je me suis demandé comment améliorer la précision de mes prédictions. Je me suis dis que j'allai améliorer ma fonction Distance() car elle ne prend pas en compte les différences de grandeurs entres les caractéristiques de l'iris (cases). L'écart de la caractéristique 1 qui est d'une grandeur supérieur aux autres influence grandement la fonction Distance() et rend négligeable les autres écarts. J'ai donc fait la moyenne des 803 individus puis diviser chaque caréristique par la plus petite caratéritique que nous avions pour connaitre les coefficients de grandeurs [25.523,5.713,1,1.74,1.36] (fonction ponderation()). J'ai alors pondéré les écarts avec les coefficients ci-dessus dans la fonction Distancepond() puis lancé de noueau graphek() pour analyser k.



Sous cette pondération nous n'observons plus de tendance. Cela ne me convenait pas, je me suis alors mis a pondérer dans Distancepond2() par rapport aux grandeurs des écarts et non plus aux grandeurs des caractéristiques même.



Je retrouve la tendance mais les k ne sont pas plus performants donc je vais étudier finalTest avec Distance() ma fonction initiale.

4) FinalTest

J'ai créé la fonction DataLoadSetfinal() afin de charger dans mon trainingSet les données connus du fichier data et preTest et dans testSet individus à prédire du fichier finalTest. A travers Mainfinal() j'ai mis mes prédictions pour les 3000 individus dans un fichier texte Ourradour_sample. J'ai vérifié par la suite mon fichier txt dans le checklabels.

```
In [217]: 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
     ...:
     ...: 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('Ourradour_sample.txt','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!")
Labels Check : Successfull!
```