20/21

RAPPORT PROJET IA : classification challenge

CHARLES DELEMAZURE

PLAN D'ACTION

Algorithme KNN

L'algorithme des k plus proches voisins ou k-nearest neighbors (kNN) est un algorithme qui à partir d'un jeu de données et d'une donnée « cible », détermine les k données les plus proches de la cible. C'est un algorithme d'apprentissage automatique (Machine Learning) supervisé simple et facile à mettre en œuvre que nous utiliserons dans notre cas pour résoudre des problèmes de classification.

Les données d'entrainement/d'apprentissage et les données cible

```
def open_file (nomdufichier):
    # On extrait les données d'entrainement de data.csv
    file = open(r"C:\Users\charl\OneDrive\Laptop\Téléchargements\{}.csv".format(nomdufichier),"r")
    data = []
    reader= csv.reader(file,delimiter=",")
    for row in reader:
        if nomdufichier != "finalTest" :
            data.append({"d0":float(row[0]),
                         "d1":float(row[1]),
                         "d2":float(row[2]),
                         "d3":float(row[3]),
                         "d4":float(row[4]),
                         "d5":float(row[5]),
                         "classe":row[6]})
        else :
            data.append({"d0":float(row[0]),
                         "d1":float(row[1]),
                         "d2":float(row[2]),
                         "d3":float(row[3]),
                         "d4":float(row[4]),
                         "d5":float(row[5])})
    return data
```

Pour d'abord récupérer les données des fichiers csv j'utilise ma fonction open_file(nomdufichier) qui prend en paramètre le nom du fichier csv. S'il s'agit du fichier finalTest.csv la colonne classe n'existe pas et n'est pas prise en compte. Autrement la fonction retourne une liste d'ensembles, chaque ensemble étant modélisé par un dictionnaire dont les valeurs des clés sont les données de chaque ligne du fichier csv séparées par des virgules (delimiter= « , »).

Proximité entre les données

Pour prédire la classification d'une nouvelle donnée, l'algorithme se base sur les k enregistrements issus de l'ensemble de données d'apprentissage et sont alors localisés les plus similaires à ce nouvel enregistrement.

L'algorithme repose sur la notion de « proximité » entre données. La similitude entre les enregistrements peut être mesurée de différentes manières. Cette proximité est mesurée à l'aide d'une distance.

Nous pouvons utiliser :

- Distance euclidienne donnée par la formule : $\text{distance}(\text{donnée1}, \text{donnée2}) = \sqrt{(x1\text{-}x2)^2\text{-}(y1\text{-}y2)^2}$

```python
def distance_euclidienne(data1,data2):
    """
    Renvoie la distance euclidienne entre deux ensembles
    """
    return sqrt((data2["d0"]-data1["d0"])**2+
                (data2["d1"]-data1["d1"])**2+
                (data2["d2"]-data1["d2"])**2+
                (data2["d3"]-data1["d3"])**2+
                (data2["d4"]-data1["d4"])**2+
                (data2["d5"]-data1["d5"])**2)
```

- Distance de Manhattan donnée par la formule : $\text{distance}(\text{donnée1}, \text{donnée2}) = |x1-x2| + |y1-y2|$

```python
def distance_manhattan(data1,data2):
    """
    Renvoie la distance de manhattan entre deux ensembles
    """
    return abs(data1["d0"]-data2["d0"]) +
           abs(data1["d1"]-data2["d1"]) +
           abs(data1["d2"]-data2["d2"]) +
           abs(data1["d3"]-data2["d3"]) +
           abs(data1["d4"]-data2["d4"]) +
           abs(data1["d5"]-data2["d5"])
```

- Distance de Tchebychev donnée par la formule : $\text{distance}(\text{donnée1}, \text{donnée2}) = \max(|x1-x2|, |y1-y2|)$

```python
def distance_tchebychev(data1,data2):
    """
    Renvoie la distance de Tchebychev entre deux ensembles
    """
    return max(data1["d0"]-data2["d0"],
               data1["d1"]-data2["d1"],
               data1["d2"]-data2["d2"],
               data1["d3"]-data2["d3"],
               data1["d4"]-data2["d4"],
               data1["d5"]-data2["d5"])
```

Nous verrons après quelle façon de calculer la distance nous prendrons.

K plus proches voisins

Il s'agit maintenant de déterminer, parmi ces distances, celles qui constituent les k plus petites. Pour cela on utilisera la fonction k_plus_proches(k,table,nouveau) :

```python
def k_plus_proches(k,table,nouveau):
    """
    Paramètre : le meilleur k, les valeurs d'entrainement, un nouvel ensemble dont on doit prédire la classe
    Résultat : une liste des k voisins les plus proches de cette ensemble
    """
    # On trie les données de la table selon la distance croissante avec la donnée cible
    # On définie le critère de trie
    def distance_nouveau(data):
        return distance_manhattan(data,nouveau)

    # On trie la table selon le critère choisi
    table_triee = sorted(table,key=distance_nouveau)

    # Prendre les k premières valeurs de la table triée
    proches_voisins = []
    for i in range(k): #range(1,k+1) lorsque l'on compare avec le meme dataset sans la colonne des classes
        proches_voisins.append(table_triee[i])

    # On renvoie les plus proches voisins
    return proches_voisins
```

Cette fonction prend en paramètre le k qu'on aura choisi, les valeurs d'entrainement dont on connaît les classes et le nouvel ensemble dont on doit prédire la classe. Dans cette fonction, on crée la fonction distance_nouvea(data) qu'on utilise avec la fonction sorted(). Cela nous permet de trier tous les ensembles dont on connait les classes en fonction de la distance avec le nouvel ensemble dont on doit déterminer la classe. La fonction k_plus_proches(k,table,nouveau) nous renverra ainsi les k voisins les plus proches.

Pour sélectionner la valeur de k qui convient à nos données, nous exécutons plusieurs fois l'algorithme KNN avec différentes valeurs de k. Puis nous choisissons le k qui réduit le nombre d'erreurs rencontrées tout en maintenant la capacité de l'algorithme à effectuer des prédictions avec précision lorsqu'il reçoit des données nouvelles (non vues auparavant).

Pour cela j'ai créé une fonction compare(train,test,k) pour les fichiers data.csv et preTest.csv. Cette fonction prend en paramètre l'ensembles des données des derniers fichiers, le k choisi et comme nouveaux ensembles les données des derniers fichiers (data, preTest) sans la colonne des classes (que l'on va prédire).

```
def compare (train,test,k):
    satisfaction = 0
    for i in range(len(test)):
        if (train[i]["classe"] == attribution(k,train,test[i])):
            satisfaction += 1
    taux = (satisfaction/len(test))*100
    print ("Pour k = ", k ," taux de satisfaction : " , taux,"%")
```

Pour tester les différentes valeurs de k à l'aide de cette fonction on devait remplacer dans la fonction k_plus_proche(k,table,nouveau) :

For i in range(k) → for i in range(1,k+1)

Sinon le voisin le plus proche serait le même ensemble à prédire la classe.

<u>Test effectués</u>

Le meilleur k pour les valeurs de data.csv en utilisant la fonction compare(train,test,k) avec différentes façons de calculer la distance entre les données :

```
Pour k= 1  taux de satisfaction :  90.0373599003736 %
Pour k= 2  taux de satisfaction :  90.0373599003736 %
Pour k= 3  taux de satisfaction :  91.03362391033623 %
Pour k= 4  taux de satisfaction :  91.03362391033623 %
Pour k= 5  taux de satisfaction :  91.15815691158157 %
Pour k= 6  taux de satisfaction :  91.03362391033623 %
Pour k= 7  taux de satisfaction :  91.03362391033623 %
Pour k= 8  taux de satisfaction :  90.28642590286425 %
Pour k= 9  taux de satisfaction :  90.66002490660024 %
Pour k= 10  taux de satisfaction :  90.16189290161893 %
Pour k= 11  taux de satisfaction :  89.91282689912828 %
Pour k= 12  taux de satisfaction :  89.53922789539229 %
Pour k= 13  taux de satisfaction :  89.29016189290162 %
Pour k= 14  taux de satisfaction :  88.66749688667497 %
Pour k= 15  taux de satisfaction :  88.66749688667497 %
```

Distance de Manhattan

```
Pour k =  1  taux de satisfaction :  89.29016189290162 %
Pour k =  2  taux de satisfaction :  89.29016189290162 %
Pour k =  3  taux de satisfaction :  90.78455790784558 %
Pour k =  4  taux de satisfaction :  91.15815691158157 %
Pour k =  5  taux de satisfaction :  91.2826899128269 %
Pour k =  6  taux de satisfaction :  90.41095890410958 %
Pour k =  7  taux de satisfaction :  90.78455790784558 %
Pour k =  8  taux de satisfaction :  89.53922789539229 %
Pour k =  9  taux de satisfaction :  88.91656288916563 %
Pour k =  10  taux de satisfaction :  89.1656288916563 %
Pour k =  11  taux de satisfaction :  88.91656288916563 %
Pour k =  12  taux de satisfaction :  88.4184308841843 %
Pour k =  13  taux de satisfaction :  87.79576587795765 %
Pour k =  14  taux de satisfaction :  87.67123287671232 %
Pour k =  15  taux de satisfaction :  88.04483188044831 %
```

Distance Euclidienne

```
Pour k =  1  taux de satisfaction :  73.22540473225405 %
Pour k =  2  taux de satisfaction :  73.22540473225405 %
Pour k =  3  taux de satisfaction :  72.85180572851806 %
Pour k =  4  taux de satisfaction :  73.10087173100872 %
Pour k =  5  taux de satisfaction :  72.35367372353674 %
Pour k =  6  taux de satisfaction :  72.47820672478207 %
Pour k =  7  taux de satisfaction :  70.9838107098381 %
Pour k =  8  taux de satisfaction :  71.3574097135741 %
Pour k =  9  taux de satisfaction :  71.48194271481943 %
Pour k =  10  taux de satisfaction :  71.48194271481943 %
Pour k =  11  taux de satisfaction :  71.85554171855541 %
Pour k =  12  taux de satisfaction :  71.48194271481943 %
Pour k =  13  taux de satisfaction :  71.73100871731009 %
Pour k =  14  taux de satisfaction :  71.60647571606475 %
Pour k =  15  taux de satisfaction :  70.85927770859277 %
```

Distance de Tchebychev

Premièrement nous pouvons remarquer que le taux de satisfaction en utilisant la distance de Tchebychev est très faible nous n'utiliserons donc pas cette méthode pour calculer la distance entre les données.

Pour le fichier data.csv nous pouvons retenir que le k le plus efficace pour prédire les classes de data.csv est k = 4 en utilisant la distance de Manhattan et Euclidienne. On regarde maintenant avec le fichier preTest.csv :

```
Pour k= 1  taux de satisfaction :  89.66376089663761 %
Pour k= 2  taux de satisfaction :  89.66376089663761 %
Pour k= 3  taux de satisfaction :  88.7920298879203 %
Pour k= 4  taux de satisfaction :  90.0373599003736 %
Pour k= 5  taux de satisfaction :  88.54296388542964 %
Pour k= 6  taux de satisfaction :  89.41469489414695 %
Pour k= 7  taux de satisfaction :  88.4184308841843 %
Pour k= 8  taux de satisfaction :  89.53922789539229 %
Pour k= 9  taux de satisfaction :  88.66749688667497 %
Pour k= 10  taux de satisfaction :  89.41469489414695 %
Pour k= 11  taux de satisfaction :  88.7920298879203 %
Pour k= 12  taux de satisfaction :  89.78829389788294 %
Pour k= 13  taux de satisfaction :  88.91656288916563 %
Pour k= 14  taux de satisfaction :  89.1656288916563 %
Pour k= 15  taux de satisfaction :  87.67123287671232 %
```

Distance de Manhattan

```
Pour k =  1  taux de satisfaction :  88.66749688667497 %
Pour k =  2  taux de satisfaction :  88.66749688667497 %
Pour k =  3  taux de satisfaction :  89.78829389788294 %
Pour k =  4  taux de satisfaction :  89.91282689912828 %
Pour k =  5  taux de satisfaction :  88.16936488169364 %
Pour k =  6  taux de satisfaction :  88.16936488169364 %
Pour k =  7  taux de satisfaction :  88.4184308841843 %
Pour k =  8  taux de satisfaction :  88.29389788293898 %
Pour k =  9  taux de satisfaction :  87.42216687422167 %
Pour k =  10  taux de satisfaction :  88.29389788293898 %
Pour k =  11  taux de satisfaction :  87.67123287671232 %
Pour k =  12  taux de satisfaction :  88.54296388542964 %
Pour k =  13  taux de satisfaction :  87.79576587795765 %
Pour k =  14  taux de satisfaction :  88.16936488169364 %
Pour k =  15  taux de satisfaction :  87.17310087173101 %
```

Distance Euclidienne

Pour le fichier preTest.csv nous pouvons retenir que le k le plus efficace pour prédire les classes de data.csv est k = 5 en utilisant la distance de Manhattan et Euclidienne. Enfin on regarde maintenant en fusionnant les deux fichiers csv :

```
Pour k= 1  taux de satisfaction :  89.8505603985056 %
Pour k= 2  taux de satisfaction :  89.8505603985056 %
Pour k= 3  taux de satisfaction :  89.91282689912828 %
Pour k= 4  taux de satisfaction :  90.53549190535492 %
Pour k= 5  taux de satisfaction :  89.8505603985056 %
Pour k= 6  taux de satisfaction :  90.2241594022416 %
Pour k= 7  taux de satisfaction :  89.72602739726028 %
Pour k= 8  taux de satisfaction :  89.91282689912828 %
Pour k= 9  taux de satisfaction :  89.66376089663761 %
Pour k= 10  taux de satisfaction :  89.78829389788294 %
Pour k= 11  taux de satisfaction :  89.35242839352429 %
Pour k= 12  taux de satisfaction :  89.66376089663761 %
Pour k= 13  taux de satisfaction :  89.10336239103363 %
Pour k= 14  taux de satisfaction :  88.91656288916563 %
Pour k= 15  taux de satisfaction :  88.16936488169364 %
```

Distance de Manhattan

```
Pour k =  1  taux de satisfaction :  88.97882938978829 %
Pour k =  2  taux de satisfaction :  88.97882938978829 %
Pour k =  3  taux de satisfaction :  90.28642590286425 %
Pour k =  4  taux de satisfaction :  90.53549190535492 %
Pour k =  5  taux de satisfaction :  89.72602739726028 %
Pour k =  6  taux de satisfaction :  89.29016189290162 %
Pour k =  7  taux de satisfaction :  89.60149439601494 %
Pour k =  8  taux de satisfaction :  88.91656288916563 %
Pour k =  9  taux de satisfaction :  88.16936488169364 %
Pour k =  10  taux de satisfaction :  88.72976338729764 %
Pour k =  11  taux de satisfaction :  88.29389788293898 %
Pour k =  12  taux de satisfaction :  88.48069738480697 %
Pour k =  13  taux de satisfaction :  87.79576587795765 %
Pour k =  14  taux de satisfaction :  87.92029887920299 %
Pour k =  15  taux de satisfaction :  87.60896637608965 %
```

Distance Euclidienne

En fusionnant les deux fichiers nous pouvons retenir que le k le plus efficace pour prédire les classes est k=4 en utilisant la distance de Manhattan et Euclidienne.

En prenant l'ensemble des taux de satisfactions pour chaque fichier, nous pouvons remarquer que la distance de Manhattan nous permet d'obtenir un meilleur taux de satisfaction en général. Nous utiliserons donc la distance de Manhattan pour faire nos prédictions.

Trouver la classe majoritaire

Pour trouver la classe majoritaire d'une liste d'ensembles prenant en paramètre cette dernière et renvoyant la classe majoritaire on utilise une fonction auxiliaire :

```python
def frequence_classe (table):
    """
    Paramètre : une liste d'ensembles de 5 variables, chaque ensemble étant modélisé par un dictionnaire
    Résultat : Un dictionnaire dont les clés sont les classes et les valeurs, le nombre de fois où cette classe apparait
    """
    frequence = {}
    for data in table:
        classe = data["classe"] #on prend la classe de chaque ensemble
        if classe in frequence.keys(): #si cette la clé de cette classe est déjà créée
            frequence[classe] = frequence[classe] + 1 #on incrémente de 1 le nombre de fois qu'elle apparait
        else:
            frequence[classe] = 1 #on crée une clé dans le dictionnaire frequence pour chaque classe qu'on initialise à 1
    return frequence
```

La fonction frequence_classe(table) prend en paramètre une liste d'ensembles et permet de retourner un dictionnaire dont les clés sont les classes et les valeurs le nombre de fois que cette classe apparaît.

```python
def classe_majoritaire (table):
    """
    Paramètre : une liste d'ensembles de 5 variables, chaque ensemble étant modélisé par un dictionnaire
    Résultat : le nom de la classe la plus représentée dans cette liste
    """
    frequences = frequence_classe(table) #on compte le nombre de fois que chaque classe apparaît
    classe_max = table[0]["classe"] #on initie la classe_max par une valeur par défaut
    for (classe,nombre) in frequences.items():
        if frequences[classe_max] < nombre :
            classe_max = classe #on prend la classe qui appraît le plus de fois
    return classe_max
```

On utilise ensuite la fonction classe_majoritaire(table) qui prend le dictionnaire retourné par la fonction frequence_classe(table) et retourne la classe qui a la valeur la plus grande.

Prédiction

```python
def attribution (k,table,nouveau):
    """
    Paramètre : le meilleur k, les valeurs d'entrainement, un nouvel ensemble dont on doit prédire la classe
    Résultat : la classe prédite du nouvel ensemble
    """
    voisins = k_plus_proches(k, table, nouveau) #on prend les k voisins les plus proches de l'ensemble
    classe = classe_majoritaire(voisins) #on prend la classe la plus fréquente de ces k voisins pour prédire la classe de l'ensemble
    return classe
```

La fonction attribution(k,table,nouveau) prend en paramètre un ensemble dont on doit prédire la classe, le k choisi et les valeurs d'apprentissages.

Cette fonction va appeler la fonction k_plus_proches() pour obtenir les k voisins les plus proches par rapport au nouvel ensemble. Puis parmi ces k plus proches voisins on appelle la fonction classe_majoritaire() pour trouver la classe la plus fréquente parmi ces voisins et enfin prédire la classe du nouvel ensemble.

Lissage des valeurs

Nous pouvons remarquer que lorsque l'on prédit les classes du fichier finalTest.csv avec différentes données d'apprentissage la prédiction peut parfois varier. Ici par exemple on peut voir différentes prédictions de deux ensembles différents en prenant en premier le fichier data.csv comme les données d'apprentissages puis preTest.csv puis les deux fusionnés.

```
prediction avec data :          classA
prediction avec pretest :       classB
prediction avec data+pretest :  classB
prediction finale -->  classB
```

```
prediction avec data :          classB
prediction avec pretest :       classA
prediction avec data+pretest :  classA
prediction finale -->  classA
```

Pour fusionner les deux fichiers csv j'ai créé une fonction unify(data1,data2) pour pouvoir fusionner deux listes retournées par la fonction open_file(nomdufichier).

```python
def unify (data1,data2):
    """
    Paramètre : deux listes d'ensembles dont chaque ensemble est modélisé par un dictionnaire
    Résultat : une liste ayant fusionné les deux listes prises en paramètre
    """
    unify = []
    for row in data1:
        unify.append(row)
    for row in data2:
        unify.append(row)
    return unify
```

On utilise donc la fonction get_prediction(train,test,k) qui prend en paramètre le k choisi, les valeurs d'entrainement et les nouveaux ensembles dont on doit prédire la classe.

```python
def get_prediction(train,test,k):
    """
    Paramètre : le meilleur k, les valeurs d'entrainement, les nouveaux ensembles dont on doit prédire la classe
    Résultat : une liste des prédictions, chaque prédictions étant modélisé par un dictionnaire
    """
    prediction = [] #création de la liste des prédictions
    for i in range(len(test)): #on parcours les nouveaux ensembles
        dico = {}
        dico["classe"] = attribution(k,train,test[i]) #chaque prédiction est modélisé par un dictionnaire
        prediction.append(dico) #pour pouvoir ensuite utiliser la fonction classe_majoritaire
    return prediction
```

Cette fonction retourne la liste des prédictions dont chaque prédiction est modélisée par un dictionnaire. On choisit d'utiliser ce format pour ensuite utiliser la fonction classe_majoritaire() et choisir la classe la plus fréquente parmi les 3 prédictions utilisant des valeurs d'apprentissages différentes (data.csv, preTest.csv, data.csv + preTest.csv).

```python
def final_prediction(data,preTest,both):
    """
    Paramètre : Les predictions du fichier finalTest avec pour valeur d'entrainement data puis preTest puis data+preTest
    Résultat : Ecrit dans un fichier text le lissage de ces prédictions
    """
    #Creation et ecriture du fichier txt
    myfile = open(r"C:\Users\charl\OneDrive\Laptop\Téléchargements\delemazure.txt", "w+")
    # On parcourt les predictions
    for i in range(len(data)):
        liste = [] # on regroupe les 3 predictions dans une liste pour pouvoir utiliser la fonction classe_majoritaire
        liste.append(data[i])
        liste.append(preTest[i])
        liste.append(both[i])
        classe = classe_majoritaire(liste) # parmis les 3 prédictions faites au dessus on prend la plus fréquente
        if i == 0 : #pas de saut de ligne pour la première prédiction
            myfile.write(classe) #ecriture de la prediction dans le fichier txt
        else:
            myfile.write("\n" + classe) #ecriture de la prediction dans le fichier txt
    myfile.close() #fermeture du fichier txt
```

Enfin on utilise la fonction final_prediction() qui prend en paramètre les prédictions retournés par la fonction get_prediction() pour les différentes valeurs d'apprentissages. Pour chaque ensemble de finalTest.csv on crée une liste regroupant les prédictions de classe de cet ensemble sous forme de dictionnaire. Cela nous permet ensuite d'appeler la fonction classe_majoritaire() pour choisir la fonction la plus fréquente parmi les trois.

Dans cette fonction on déclare notre nouvelle variable myfile puis on utilise les commandes d'ouverture et d'écriture intégrées pour ouvrir et écrire dans le fichier. Le «w +» indique à Python que nous allons écrire un nouveau fichier. Si le fichier existe déjà, cela écrasera le fichier. Si on le remplace par un «w», le fichier sera créé uniquement s'il n'existe pas déjà. On utilise myFile.write pour écrire les prédictions dans le fichier txt. Notons que nous devons toujours fermer le fichier à la fin pour que les modifications soient enregistrées.

Zone du main

```
"""ZONE DU MAIN"""

if __name__ == '__main__':
    start_time = time.time()

    #Ouverture des fichiers
    data = open_file("data")
    preTest = open_file("preTest")
    both = unify(data,preTest)
    test = open_file("finalTest")
    #Predictions
    pred_data = get_prediction(data,test,5)
    pred_preTest = get_prediction(preTest,test,4)
    pred_both = get_prediction(both,test,4)
    #On choisis la prédiction la plus fréquente parmis les 3
    final_prediction(pred_data,pred_preTest,pred_both)


    print("\nTemps d'exécution : %s seconds" % round((time.time() - start_time),4))
```

Précédemment nous avons pu remarquer que chaque fichier avait un k plus ou moins efficace. Ainsi on utilise le meilleur k pour chaque prédiction en fonction des valeurs d'apprentissage :

- Le fichier data.csv avait un meilleur k égale à 5.
- Le fichier preTest.csv avait un meilleur k égale à 4.
- Les deux fusionnés avait un meilleur k égale à 4.

Le programme prend exactement 18.2143 secondes à s'effectuer.

Les imports

On importe :

- **Csv :** pour lire les fichiers csv
- **Time :** pour compter le temps d'exécution du programme