Aymeric Chazottes, Djallel Dilmi, Cécile Mallet, Sylvie Thiria

TP mise en î uvre : Algorithme des k plus proches voisins (kppv)

I - Les Objectifs

Le but de ce TP est de prendre en main l'algorithme des k plus proches voisins. Il est Organisé en plusieurs parties.

ŏ La première partie consiste à optimiser la classification sur des ensembles simulés par Kpp. On étudiera :

- L'importance / impact du nombre de plus proches voisins considérés.
- L'importance / impact du choix de la distance utilisée pour comparer deux points.
- L'importance / impact du choix codage utilisé pour le vote de la décision.
- L'importance / impact de la représentativité de l'échantillon d'apprentissage sur la frontière de décision.

ŏ Dans la seconde partie on opposera la classification par kpp à la classification par réseau de neurones avec une application sur les iris de Fisher. On étudiera ainsi :

- L'importance / impact du choix de l'algorithme de classification.

ŏ Dans la troisième partie on opposera aussi la classification par kpp à la classification par réseau de neurones avec une application sur la classification des chiffres manuscrits. On étudiera aussi :

- L'importance / impact du choix codage utilisé sur les performances des algorithmes.

II - Eléments pour la réalisation du TP

Pour la réalisation de ce TP, il est fortement recommandé d'utiliser les bibliothèques standards de python et de ne rien coder dans la mesure du possible. Toutes les méthodes nécessaires se trouvent dans la suite **scipy (pandas, numpy, scikit-learn, matplotlib,…)**

III - 1ère Partie du TP :

1°) **Expliquer le principe de fonctionnement, la nature et les différentes étapes d'un algorithme des k plus proches voisins**, puis retrouver si possible les différentes étapes spécifiquement associées à cet algorithme dans la librairie Scikit-learn[1].

2°) **On dispose maintenant d'un ensemble de 132 données d'apprentissage en dimension 2, labellisées en 3 classes d'égale taille.** Les données sont contenues dans le fichier **DATA1_app.txt** avec les étiquettes représentées sur la troisième colonne. Avec ces données d'apprentissage, le second script classe les 99 données contenues dans **DATA1_test.txt** avec algorithme des k plus proches voisins. Les données à classer ont été étiquetées par un expert, les labels correspondants sont contenus sont représentées sur la troisième colonne du fichier. Les labels sont utilisés pour analyser les performances des modèles (ex. calculer la matrice de confusion (MCO)).
Analyser les résultats obtenus pour différentes valeurs de k (1, 2, 3, 4, 5, 10, 15, 20) **et différentes distances utilisées et le système de poids considérés lors de la prise de décision** (faire un tableau).
En particulier, pour chacune des distances, expliquer les différents cas de figures de classification obtenues pour le meilleur et le moins bon des résultats ainsi que la matrice de confusion.

Partie II : k plus proches voisins appliqués aux iris de Fisher

Les Iris de Fisher est un exemple classique de classification. Pour rappel, le jeu de données (vu dans l'U.E. ACP) contient 150 exemples d'iris décrites par 4 variables qui sont : la hauteur et la largeur du sépale (HS et LS), la hauteur et la largeur du pétale (HP et LP). L'expert (Mr Fisher) a identifié 3 classes pour ces iris dénommées Sétosa, Versicolor et Virginica. Ces classes sont indiquées par les indices 1, 2 ou 3 dans le jeu de données qui est présent dans les datasets de plusieurs libraires dont scikit-learn.
1) Si cela n'a pas été fait faire une étude préliminaire du jeu de données.
Représenter les données en 2D en utilisant une ACP et une T-SNE.
2) Pour la suite de l'exercice, il faut séparer l'ensemble des données en un ensemble d'apprentissage et un ensemble de test. On vous demande de réaliser les expériences décrites ci-après (point 3) en envisageant les 3 cas de répartition suivants : 1/3, 1/2 puis 2/3 en apprentissage et le reste en test.

[1] Cette étape ne doit pas vous prendre plus de 10 minutes sinon passez à l'étape suivante.

3) Pour chacun de ces cas :

a) Représenter l'ensemble de test sur le plan de représentation choisi dans la question 1 (individus supplémentaires).

b) Classification par l'algorithme des k plus proches voisins.

- Appliquer l'algorithme des k plus proche voisins pour différentes valeurs de k (k=2, 3, 4, 5, 6, 7, 10, par exemple), et en utilisant seulement la distance euclidienne en utilisant les 4 variables d'une part et le plan 2D de représentation choisi dans la question 1. Commenter les résultats.
- Pour chaque valeur de k, calculer la performance sur cet ensemble. Ces performances devront être mentionnées dans le rapport.
- Pour la meilleure performance obtenue sur l'ensemble de test :
  - Donner la matrice de confusion.
  - Présenter la figure de cet ensemble qui fait ressortir les points mal classés.

Partie III : k plus proches voisins appliqués à la reconnaissance des chiffres Manuscrits

Le but de cette partie est de reconnaître des chiffres manuscrits à l'aide des Kpp. Elle Comprend deux parties :

ŏ La première partie consiste à optimiser la classification des chiffres manuscrits par Kpp. On étudiera :

- L'importance / impact du nombre de plus proches voisins considérés.
- L'importance / impact du choix codage utilisé pour la comparaison.

ŏ Dans la seconde partie on comparera les performances du modèle choisi avec les performances du modèle sélectionné lors du TP précédents sur les MLP et Poids partagés.

Rappel sur les Données

Pour la réalisation du TP, nous mettons à disposition les mêmes données que la dernière fois :

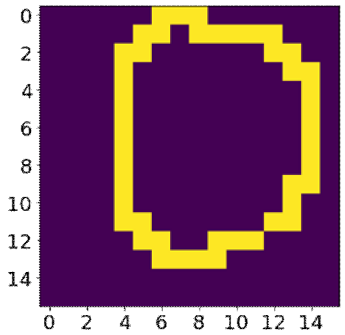
x.txt : Est la base de données de chiffres manuscrits (images). Elle est composée de 480 chiffres codés en binaire (±1), dans une matrice 256x480. Cela signifie que chaque «bitmap» binaire 16x16 a été transformée en un vecteur de dimension 256 qui, à son tour, correspond à une colonne de la matrice du fichier **x.txt**.

t.txt : Contient les sorties désirées qui sont associées aux chiffres de **x.txt** dans le même ordre. Chaque réponse désirée est le chiffre correspondant.

visualisation des données :

- Après avoir chargé les données, utiliser la fonction **reshape** du package **numpy** pour retrouver la forme carrée de l'image.
- Ensuite Utiliser la fonction **imshow** associée au **matplotlib.pyplot** pour la visualisation de la base de données. Par exemple pour voir la première forme :

```
>>first_image=np.reshape(x[:,0],(16,16))
# pour l'image 16x16=256
>> plt.imshow(fisrt_image)
```



III - Les différents codages utilisés :

Les deux méthodes (kpp et kmeans) utilisent des distances. Le calcul des distances entre les objets est parfois complexe et peut prendre du temps sans apporter une information pertinente. Pour réduire cette complexité d'une part et ne conserver que l'information nécessaire pour la classification, il est d'usage de coder l'information (la résumer) pour simplifier le calcul des distances.

L'un des objectif de ce TP est de comparer les performances de la classification en fonction des différents types de codage que l'on va faire sur les chiffres, et qui serviront d'entrée pour le calcul des distances.

Description des codages utilisés :

HX : histogramme des projections du chiffre sur l'axe des x (dans chaque colonne on compte le nombre de pixels noir). HX est donc un vecteur de 16 composantes.

HY : histogramme des projections du chiffre sur l'axe des y (dans chaque ligne on compte le nombre de pixels noir). HY est aussi un vecteur de 16 composantes.

PH : profil haut – pour chaque colonne, on code la coordonnée de la première transition blanc/noir en partant du haut. PH est un vecteur de 16 composantes.

PB : profil bas – pour chaque colonne, on code la coordonnée de la première transition blanc/noir en partant du bas. PB est un vecteur de 16 composantes.

PG : profil gauche – pour chaque ligne, on code la coordonnée de la première transition blanc/noir en partant de la gauche. PG est un vecteur de 16 composantes.

PD : profil droit – pour chaque ligne, on code la coordonnée de la première transition blanc/noir en partant de la droite. PD est un vecteur de 16 composantes.

Les différents fichiers correspondant aux différents codages

Image sous forme d'un vecteur de 256 pixels (codage utilisée dans la première partie). Fichier **caract.mat**.

(HX,HY), vecteur de 32 composantes. Fichier d'entrée : **hx_hy.txt** ;

(PG,PD), vecteur de 32 composantes. Fichier d'entrée : **pg_pd.txt** ;

(HX,HY,PG,PD), vecteur de 64 composantes. Fichier d'entrée : **hx_hy_pg_pd.txt** ;

(PB,PH), vecteur de 32 composantes. Fichier d'entrée : **pb_ph.txt** ;

(HX,HY,PB,PH), vecteur de 64 composantes. Fichier d'entrée : **hx_hy_pb_ph.txt** ;

Après avoir compris les différents codages, on indiquera avec visualisation à l'appui l'intérêt de chacun des codages. Pour cela utiliser la fonction **plot** associée aux **matplotlib.pyplot**

Réalisation du TP :

En utilisant la classe **KNeighborsClassifier** associée au package **sklearn.neighbors**, optimiser la classification des chiffres manuscrits.

Rappel des étapes :

1) Choix du nombre de plus proches voisins : pour chaque type de codage, vous devrez créer puis entrainer un classifieur de type kpp et chercher un nombre « optimal » de voisins à considérer en faisant varier k lors de la construction du classifieur.

- Utiliser le constructeur **KNeighborsClassifier** associée au package **sklearn.neighbors** pour créer les différents classifieurs (différents k).
- Utiliser la méthode **fit** associée aux classifieurs crées pour l'apprentissage
  (Attention au format des entrées/sorties, un travail de préparation des données est nécessaire).
- Utiliser la méthode **score** associée aux différents classifieurs pour la comparaison des performances globales.

Quel est le nombre de plus proches voisins « optimal » à considérer pour chaque type de codage?

2) Impact du codage utilisé : comparer les performances des classifieurs obtenus avec différents codages.

- Analyser les résultats d'un point de vue global (% de chiffre mal classés) en utilisant la méthode **score** associée aux classifieurs, et plus finement en étudiant les matrices de confusion en utilisant la méthode **confusion_matrix** associée à **sklearn.metrics**.

Suivant les codages : certains chiffres sont-ils plus faciles à classer que d'autres ? Si oui, à votre avis pourquoi ? Justifier votre réponse.