TP D'INFORMATIQUE N°5
Algorithme des k plus proches voisins

1 Implémentation de knn

Dans la suite, on identifiera un point au tuple de ses coordonnées, et on appelera *point annoté* un couple formé d'un point et d'une étiquette (entière).

1. Écrire une fonction `distance` prenant en argument deux points ayant le même nombre de coordonnées, et renvoyant la distance euclidienne entre ces points.
2. On veut à présent écrire une fonction `liste_voisins` prenant en argument un point `p`, un entier `k`, et un échantillon d'entraînement (ie une liste de points annotés) `Ltrain`, et renvoyant la liste des k points annotés les plus proches du point p.

   Implémenter une version de cette fonction utilisant la fonction `sorted` de Python : `sorted(L, key = f)` renvoie une version de la liste `L` triée par valeurs croissantes des `f(x)`, `x` $\in$ `L`. On prendra bien soin de définir préalablement la bonne fonction `f` à l'intérieur de la définition de `liste_voisins`. On notera bien que cette fonction `f` doit prendre en argument un élément de `L`, c'est-à-dire un point annoté.
3. Écrire une fonction `etiquette_maj` prenant en argument une liste de points annotés `V`, et renvoyant une étiquette de nombre d'occurrences maximal dans `V`.
4. Écrire une fonction `knn` prenant en argument un point `p`, un entier `k` et un échantillon d'entrainement `Ltrain`, et renvoyant l'étiquette prédite par l'algorithme des k plus proches voisins pour le point `p`.

2 Import des données annotées

Pour tester la fonction `knn`, il faut disposer de données annotées. Chacun des fichiers `L1.txt` et `L2.txt` disponibles sur cahier de prepa contient une liste de points annotés, sous la forme d'un point annoté `x,y,e` par ligne.

1. Écrire une fonction `fichier_vers_liste` prenant en argument le nom d'un tel fichier texte, et renvoyant la liste de points annotés correspondante.

   On redonne les fonctions suivantes pour la lecture de fichier :
   - `f=open('nom_du_fichier.txt','r')` permet d'ouvrir un fichier en lecture (*read*), sous le nom de variable `f`.
   - `f.readlines()` renvoie la liste des lignes présentes dans le fichier `f`, chaque ligne étant représentée sous forme de chaîne de caractères.
   - `ch.strip()` renvoie la chaîne de caractère obtenue à partir de la chaîne `ch` en supprimant tous les caractères spéciaux, tels que les retours à la ligne.
   - `ch.split(',')` renvoie la liste des sous-chaînes séparées par une virgule apparaissant dans la chaîne `ch`.
   - `f.close()` ferme le fichier `f` (ce qu'il faut toujours faire une fois qu'on a fini de travailler sur un fichier qu'on a ouvert).
2. Importer `matplotlib.pyplot` et, pour chaque étiquette présente dans `L1`, afficher avec `plot` les points annotés par cette étiquette, puis ouvrir la fenêtre graphique avec `show()`.
3. Importer la bibliothèque `random`, et utiliser la fonction `shuffle` de cette bibilothèque pour mélanger `L1` et `L2`. Former ensuite une liste `L1train` formée des 80% premiers éléments de `L1`, et `L1test` formée des éléments restants, et faire de même pour `L2`.
4. Tester la fonction `knn` à l'aide de `L1train` et `L2train` sur des points de `L1test` et `L2test`, en choisissant différentes valeurs de `k`.

3 Calibrage

1. Pour chaque valeur de k de 1 à 40, afficher le taux d'erreur empirique sur `L1test` de `knn` exploitant `L1train`. Proposer une valeur de k permettant de minimiser ce taux d'erreur. Faire de même avec `L2`.
2. Compléter l'affichage précédent pour afficher également la matrice de confusion dans chaque cas.
3. Déterminer une valeur de k permettant de minimiser la somme pour chaque point de `L2test` de $|e_c - e_p|$, où e_c est l'étiquette annotée et e_p l'étiquette prédite par `knn`.

4 Variantes

1. On se propose d'écrire une autre implémentation de la fonction `Liste_voisins`, maintenant à jour une liste des k points les plus proches de p rencontrés jusque-là dans `Ltrain`.
   (a) Écrire une procédure `inserer` prenant en argument un point `p`, un point annoté `p2` et une liste de points annotés `V` supposée triée selon la distance à `p`, et insérant `p2` dans `V` de façon à la maintenir triée.
   Par exemple, si `p = (0, 1)` et `V = [((0, 0.8), 0), ((0, 1.3), 1), ((0, 2), 0)]`, à l'issue de l'appel `inserer(p, ((0.4, 1), 1), V)`, on aura comme nouvelle valeur `V = [((0, 0.8), 0), ((0, 1.3), 1), ((0.4, 1), 1), ((0, 2), 0)]`.
   (b) Implémenter une nouvelle version de `liste_voisins` partant d'une liste vide et y insérant chaque élémennt de `Ltrain`. Une fois que la liste produite aura atteint une longueur de `k`, on enlevera après chaque insertion son élément maximal avec la fonction `pop()`.
2. Déterminer la complexité des deux implémentations de `Liste_voisins`, en fonction de k et de la longueur n de `Ltrain`. On supposera que la fonction `distance` est en $O(1)$, et que la fonction `sorted` est en $O(n \log n)$.
3. Concevoir et implémenter une version de `knn` attribuant à un voison de p d'autant plus de votes qu'il est proche de p.