TD10

On s'intéresse dans ce TD à la programmation de certaines fonctions utiles dans l'algorithme des k plus proches voisins.

1. La classe majoritaire

Dans le cours, on s'est limité à deux classes donc la détermination de la classe majoritaire était rendue très simple. Ici, on considère un nombre de classe quelconque (ce qui serait le cas dans la reconnaissance de caractères)

1. Écrire une fonction `majoritaire(L:list)->int` qui prend en argument une liste `L` (chaque élément de `L` est de la forme (x, y, c) où (x, y) sont les coordonnées des points et c la classe du point) et qui renvoie la classe majoritaire de `L`.

Dans la fonction précédente, on se contente de compter le nombre de points dans les k plus proches voisins pour déterminer la classe majoritaire. On peut tenter d'améliorer cette détermination en tenant compte de la distance des points parmi ces k plus proches voisins :

— Pour on point `p` parmi les k plus proches de `x`, on lui associe un poids égal à $\dfrac{1}{1+d(x,p)}$ (plus les points `x` et `p` sont proches, plus le poids est grand)

— Pour conjecturer la classe de `x`, on cherche la classe dont la somme des poids est la plus grande

2. Écrire une fonction `majoritaire(x:tuple,dist:function,L:list)->int` qui prend en argument un point `x`, la fonction `dist` qui permet de calculer la distance entre deux points et la liste `L` des k plus proches voisins de `x` (de la même forme que précédemment) et qui renvoie la classe prédite pour `x`.

2. Éviter un tri complet

On a dans le cours utilisé la méthode `sort()` pour trier la liste des distances avant de ne conserver que les k premières. On peut également se contenter de le conserver directement que les k premières, sans trier la liste en entier :

— On commence par créer une liste `L` avec les k premiers points

— Pour chaque nouveau point `p`, s'il est à une distance inférieure à la distance maximale des points de `L`, on remplace le point de `L` à distance maximale par `p`

1. Écrire une fonction `posMax(x:tuple,L:list,dist:function)->int` qui prend en argument un point `x`, une liste de points `L` et la fonction `dist`, permettant de calculer la distance entre deux points, et qui renvoie l'indice du point de `L` qui est à la distance maximale de `x`.

2. En déduire une fonction `plusProche(x:tuple,E:list,dist:function,k:int)->int` qui prend en argument un point `x`, une liste `L` (de couples (point,classe) comme dans le cours), une fonction `dist` permettant de calculer la distance entre deux points et un entier `k` et qui renvoie la classe à associer au point `x`.

3. Comparer la complexité de cette fonction aux lignes correspondantes dans le code du cours.

3. Choisir k

1. On suppose disposer d'un ensemble E de points avec leur classe (représenté par une liste de points de la forme (x, y, c)). Comment l'utiliser pour déterminer la meilleure valeur de k ?

2. Coder la fonction permettant cette détermination de k. *On pourra utiliser la fonction* `random.shuffle(L)` *du module* `random` *qui mélange une liste sur place ainsi que la fonction* `confusion(X,Y)` *vue en cours.*