TP : k plus proches voisins

I) Préliminaires

Exercice 1 (Fonctions auxiliaires générales)
Nous allons coder plusieurs fonctions qui serviront pour l'algorithme kNN, mais qui ne sont pas l'algorithme kNN.

1) Écrire une fonction `d_carre(x, y)` qui prend en arguments deux d-uplets `x` et `y` représentant des points de $\mathbb{R}^d$ et retourne la distance entre x et y *au carré*.
On veut minorer la distance, pour ça il est inutile de calculer une racine carrée. Ici aussi, la norme euclidienne se manipule... au carré !

2) Écrire une fonction `liste_dc(x, Ly)` qui prends en arguments un d-uplet `x`, et une liste de d-uplets `Ly`. Elle retourne la liste des distances aux carrée : si on note `L` cette liste, `L[i]` contient $\|x - Ly[i]\|_2^2$.

Exercice 2 (Données d'apprentissage)
Nous allons importer les données $\mathscr{E}$ d'apprentissage.
Dans cet exercice, ce sont des points choisis aléatoirement par python dans le carré $[-1,1]^2 \subset \mathbb{R}^2$. On étiquette en rouge ($r = 1$) les points à l'intérieur du disque unité, et en bleu ($r = 0$) ceux à l'extérieur.

1) À l'aide de l'instruction « `from donnees_app_TP1 import *` », importez les fonctions contenues dans `donnees_app_TP1.py` que vous avez reçu par mail.

2) Testez les deux fonctions.

II) Algorithme kNN

Exercice 3 (Recherche des k plus proches voisins)
Si la question 2 vous bloque, contournez la temporairement en utilisant l'instruction suivante pour trier `M` selon la seconde composante : `sorted(M, key=lambda m : m[1])`

1) Écrire une fonction qui retourne l'indice du minimum dans une liste `L`.

2) On se donne une liste `L` et on considère la liste `M=[(i, L[i]) for i in range(L)]`.
Écrire une fonction `tri_partiel(M, k)` qui trie `M` définie précédemment, par ordre croissant sur la seconde composante, en s'arrêtant après avoir trié k (entier) éléments. La méthode de tri utilisé sera un tri par sélection. On pourra adapter la fonction de la question précédente.

3) Écrire une fonction `indice_voisins(x, Ly, k)` qui prend en entrée un point `x`, une liste de points `Ly` et un entier `k`, et renvoie la liste des indices des `k` voisins de `x` dans `Ly` les plus proches. On s'aidera des fonctions `liste_dc(x, Ly)` et `tri_partiel(M, k)`.

4) Écrire une fonction `labels_voisins(Lindices, labels)` qui retourne la valeur la plus fréquente dans la liste des `labels[i]` pour `i` parcourant `Lindices`.
On pourra utiliser un dictionnaire.

Exercice 4 (Algorithme kNN)
Écrire une fonction `kNN(x, Lx, labels, k)` qui renvoie la valeur de l'étiquette associée à `x` via l'algorithme des k-plus proches voisins.

Exercice 5 (Affichage du résultat)
En balayant $[-1,1]$ à l'aide de `np.linspace(-1,1,n)`, créer deux listes `Lx_essai` et `labels_essai` contenant respectivement des points de $[-1,1]^2$ et leur étiquette déterminée par `kNN`.
Afficher le résultat (les arguments de `affiche_donnees_disque` doivent être des tableaux NumPy).

III) Amélioration : $\mathscr{E}_A$ et $\mathscr{E}_T$, tests de différents k

Dans cette section, on cherche à tester notre algorithme sur une partie des données d'apprentissage $\mathscr{E}$. Il faut donc séparer ces données en deux ensemble $\mathscr{E}_A$, qui servira à l'apprentissage, et $\mathscr{E}_T$, qui servira aux tests.

Exercice 6 (Séparation des données)
Dans l'idéal, il faut faire la liste des points pour chaque étiquette, et prendre la même proportion de points pour $\mathscr{E}_T$ dans chaque étiquette.
Nous allons faire plus simple.

1) À l'aide de `np.random.shuffle`, mélanger la liste `list(range(len(Lx)))`, où `Lx` contient l'ensemble $\mathscr{E}$. C'est une fonction *en place*.

2) Construire une fonction `partage_donnees(Lx, labels, ratio)` qui retourne un quadruplet `(Lx_test, labels_test, Lx_app, labels_app)`, où une proportion `ratio` des données est dans `(Lx_test, labels_test)` et le reste dans `(Lx_app, labels_app)`.

Exercice 7 (test)
Nous allons tester la qualité de l'algorithme : d'abord un pourcentage de données test bien étiquetées, puis la matrice de confusion.

1) Écrire une fonction `prediction(Lx_app, labels_app, Lx_test, k)`, qui retourne une liste `labels_kNN` des étiquettes calculées par `kNN(x, Lx_app, labels_app, k)`.

2) Écrire une fonction `pourcentage_reussite(labels_test, labels_kNN)` qui prend en entrée les étiquettes connues `labels_test`, ainsi que les étiquettes `labels_kNN` prédies par l'algorithme kNN, et qui renvoie le pourcentage de bonnes étiquettes.

3) Modifier la fonction `obtenir_disque` du fichier `donnees_app_TP1.py` :

```
labels = [int(np.linalg.norm(x)<= 1)* (np.random.random_sample()*10 > 1)for x in Lx]
```

Cette instruction va rajouter du « bruit », des points bleus dans le disque rouge avec une probabilité de 1/10.

Faire un graphique qui affiche le pourcentage de bonnes étiquettes pour les données test en fonction de la valeur `k` pour l'algorithme kNN.

4) Écrire une fonction `confusion(labels_test, labels_kNN)` qui calcule la matrice de confusion.