Partie I L'algorithme des k plus proches voisins

Dans cette partie, on considère un jeu de 800 données bidimensionnelles (i.e. qui sont des éléments de $\mathbb{R}^2$) se trouvant dans le fichier `data1.txt`. Pour charger et représenter ces données, on peut utiliser le code suivant.

```
import numpy as np
import matplotlib.pyplot as plt

## Chargement des données ##
fichier = open("data1.txt", mode = 'r')
lst_lignes = fichier.readlines()
fichier.close()

X = np.empty([800, 2], dtype=float)
Y = np.empty(800, dtype=int)
for i in range(800):
    X[i][0] = float(lst_lignes[i].split(";")[0])
    X[i][1] = float(lst_lignes[i].split(";")[1])
    Y[i] = float(lst_lignes[i].split(";")[2])

## Affichage des données ##
plt.scatter(X[:, 0], X[:, 1], marker="o", c=Y, s=25, edgecolor="k")
plt.show()
```

Pour chaque $i \in \llbracket 0, 799 \rrbracket$, l'élément $X[i]$ est un vecteur de $\mathbb{R}^2$ et $Y[i] \in \llbracket 0, 3 \rrbracket$ est l'étiquette associée à cette donnée. Chacune des quatre classes contient exactement 200 données. Nous allons commencer par séparer nos données en deux parties avec les instructions suivantes : une pour l'entraînement et une pour les tests.

```
## Séparation des données
X_tra = np.empty([600, 2], dtype=float)
Y_tra = np.empty(600, dtype=int)
X_test = np.empty([200, 2], dtype=float)
Y_test = np.empty(200, dtype=int)

compteurs = [0, 0, 0, 0]
n_tra, n_test = 0, 0
for i in range(800):
    if compteurs[Y[i]] < 150:
        X_tra[n_tra], Y_tra[n_tra] = X[i], Y[i]
        n_tra += 1
    else:
        X_test[n_test], Y_test[n_test] = X[i], Y[i]
        n_test += 1
    compteurs[Y[i]] += 1
```

Question 1 : Écrire une fonction `dist(v1, v2) -> float` qui prend en argument deux vecteurs `v1` et `v2` de $\mathbb{R}^2$ et renvoie la distance euclidienne entre ces derniers.

Question 2 : Écrire une fonction `plus_proches_voisins(k:int, v) -> list` qui prend en argument un entier `k` et un vecteur `v` de $\mathbb{R}^2$ et renvoie une liste contenant les indices des `k` éléments de `X_tra` les plus proches de `v`.

Question 3 : Écrire une fonction `classifier(k:int, v) -> int` qui prend en argument un entier positif `k` et un vecteur `v` de $\mathbb{R}^2$ et renvoie l'étiquette attribuée à `v` par la méthode des `k` plus proches voisins appliquée avec les données d'entraînement `(X_tra, Y_tra)`.

Question 4 : Écrire une fonction `matrice_confusion(k:int)` qui prend en argument un entier `k` et qui renvoie la matrice de confusion associée à la méthode des `k` plus proches voisins appliquée avec les données d'entraînement `(X_tra, Y_tra)` et les données de test `(X_test, Y_test)`.

Remarque 1 : Vous pourrez utiliser votre fonction pour vérifier que la matrice de confusion obtenue pour `k = 3` est

$$M_3 = \begin{pmatrix} 47 & 0 & 3 & 0 \\ 1 & 49 & 0 & 0 \\ 1 & 0 & 48 & 1 \\ 0 & 1 & 2 & 47 \end{pmatrix}.$$

Question 5 : Écrire une fonction `nb_erreurs(k:int) -> int` qui prend en argument un entier `k` et qui renvoie le nombre d'erreurs obtenues en classifiant les données de test `(X_test, Y_test)` avec la méthode des `k` plus proches voisins appliquée avec les données d'entraînement `(X_tra, Y_tra)`.

Question 6 : Écrire une fonction `tracer_nb_erreurs(kmax:int) -> None` qui prend en argument un entier `kmax` et qui affiche le tracé sur un graphique de la fonction `k ↦ nb_erreurs(k)` pour `k ∈ [1, kMax]`. Quelles valeurs de l'entier `k ∈ [1, 20]` semblent les plus intéressantes dans notre cas?

Une fois que l'on a déterminé les `k` plus proches voisins $v_1, \dots, v_k$ d'un vecteur $v \in \mathbb{R}^2$, on peut introduire une pondération permettant de donner plus de poids aux voisins les plus proches lors de l'étape finale de classification. Dans la suite, on considère que le vecteur v_i est pondéré par le coefficient

$$c_i = \frac{1}{\|v - v_i\|}.$$

Question 7 : Modifier votre fonction `classifier(k:int, v) -> int` pour qu'elle tienne compte de la pondération choisie ci-dessus.

Question 8 : Comparer les résultats obtenus avec cette nouvelle méthode de classification et la précédente.

Partie II L'algorithme des k -moyennes

II.A - Un exemple avec des données bidimensionnelles

Dans cette partie, on considère un jeu de 600 données bidimensionnelles (i.e. qui sont des éléments de $\mathbb{R}^2$) se trouvant dans le fichier `data2.txt`. Pour charger et représenter ces données, on peut utiliser le code suivant.

```
## Chargement des données ##
fichier = open("data2.txt", mode = 'r')
lst_lignes = fichier.readlines()
fichier.close()

X = np.empty([600, 2], dtype=float)

for k in range(600):
    X[k][0] = float(lst_lignes[k].split(";")[0])
    X[k][1] = float(lst_lignes[k].split(";")[1])

## Affichage des données ##
plt.plot(X[:, 0], X[:, 1], 'ko')
plt.show()
```

Dans la suite, on souhaite appliquer l'algorithme des k -moyennes au jeu de données précédent.

Question 9 : Quelle valeur de k vous paraît-elle pertinente sur cet exemple?

Question 10 : Écrire une fonction `dist(d1, d2) -> float` qui prend en argument deux données $d1$ et $d2$ (vus comme des vecteurs de $\mathbb{R}^2$) et renvoie la distance euclidienne entre elles.

Question 11 : Écrire une fonction `init(k:int)` qui renvoie un tableau numpy de k éléments distincts tirés au hasard dans X .

Question 12 : Écrire une fonction `plus_proche_centre(d, tab_centres) -> int` qui prend en argument une donnée d et un tableau numpy `tabCentres` d'éléments de $\mathbb{R}^2$ et qui renvoie l'indice dans le tableau `tabCentres` du centre le plus proche de d .

Question 13 : Écrire une fonction `k_moyennes(k:int)` qui prend en argument un entier k et qui, en appliquant l'algorithme des k -moyennes au jeu de données considéré dans cette partie, renvoie un vecteur Y de sorte que $Y[j]$ indique la classe de l'élément $X[j]$.

Question 14 : En vous appuyant sur le code donné dans l'introduction de la première partie, écrire les instructions permettant d'afficher toutes les données avec une coloration indiquant la classe à laquelle elle a été affectée (par l'algorithme des k -moyennes).

II.B - Réduction du nombre de couleurs dans une image

Dans cette partie, on s'intéresse au problème de la compression d'une image via la réduction du nombre de couleurs distinctes utilisées dans cette dernière. Nous allons utiliser l'algorithme des k -moyennes pour choisir (de manière pertinente) les nouvelles couleurs utilisées dans la version compressée de l'image.

On rappelle que chaque pixel d'une image est représenté par un triplet (R, G, B) avec $(R, G, B) \in \llbracket 0, 255 \rrbracket^3$. Pour manipuler une image en Python, on la convertit en un tableau numpy avec les instructions suivantes.

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

# Chargement d'une image d'un format classique vers un tableau numpy
imgpil = Image.open("perroquet.jpg")
img = np.array(imgpil)

# Afficher une image mémorisée avec un tableau numpy
plt.imshow(img)
plt.show()

# Sauvegarde d'une image numpy vers un format classique
imgpil = Image.fromarray(img)
imgpil.save("resultat.jpg")
```

Question 15 : Écrire une fonction `nb_couleurs(img)` qui prend en argument une image au format numpy et qui renvoie le nombre de couleurs distinctes utilisées dans l'image.

Remarque 2 : Vous pourrez utiliser votre fonction pour vérifier que l'image `perroquet.jpg` contient 40 257 couleurs distinctes.

Dans la suite, on regarde chaque pixel de l'image comme un vecteur de $\mathbb{R}^3$: nous allons appliquer l'algorithme des k -moyennes à l'ensemble des pixels afin de les regrouper en k couleurs. En pratique, l'algorithme est relativement lent (car le nombre de pixels dans une image est grand), donc on arrête l'exécution de l'algorithme avant que les couleurs ne soient complètement stabilisées.

Question 16 : Écrire une fonction `k_moyennes(img, k:int, nb_iter:int)` qui prend en arguments une image numpy `img`, un entier `k` et un entier `nb_iter` et qui renvoie les `k` couleurs obtenues au bout de `nb_iter` étapes dans l'algorithme des k -moyennes.

Question 17 : Écrire une fonction `reduction(img, k:int, nb_iter:int)` qui prend en arguments une image numpy `img`, un entier `k` et un entier `nb_iter` et qui renvoie une version modifiée de `img` n'utilisant que les `k` couleurs déterminées par la fonction de la question précédente.