Informatique du Tronc Commun

TD n°4 Intelligence artificielle

I Mise en jambe

Exercice n°1 Partition

Q1. Écrire une fonction `distribue` qui prend en paramètres un ensemble donné sous la forme d'une liste et un nombre entier `k` non nul et qui génère aléatoirement en k parties de l'ensemble contenant au moins k éléments. Utiliser pour cela un dictionnaire dont les clés sont les éléments de l'ensemble. Les valeurs sont les numéros des parties de 1 à k affectées aléatoirement à chaque élément. Toutes les valeurs doivent être affectées.

Q2. Écrire une fonction `partitionne` qui prend en paramètres un ensemble donné sous la forme d'une liste et un nombre entier `k` non nul et qui renvoie la partition en k parties de l'ensemble contenant au moins k éléments générée aléatoirement à l'aide de la fonction distribue. La fonction renvoie une liste de listes qui représentent les éléments d'une partie.

Exercice n°2 Algorithme sans tri

Dans cet exercice, pour effectuer la recherche des k plus proches voisins de x, on n'utilise aucun algorithme de tri mais on effectue un parcours séquentiel de l'ensemble.

Voici un exemple d'algorithme où on construit une liste appelée voisins qui contient les k plus proches voisins d'un point x parmi les éléments d'un ensemble `E` représenté par une liste :

```
Pour i allant de 0 à k-1
        Placer le point E[i] dans la liste voisins
Pour i allant de k à n-1
        Si la distance entre x et E[i] est inférieure à la distance x et un point de la liste voisins
                Supprimer de la lise voisins le point le plus éloigné de x en le remplaçant dans la liste voisins par le point E[i]
```

Le coût en fonction de n est linéaire si k est « très petit» devant n.

Q1. Définir une fonction `distance` qui renvoie le carré de la distance euclidienne entre deux points du plan représentés par les listes de leurs coordonnées.

Q2. Écrire une fonction `kppv` qui prend en paramètres un ensemble `E` représentant une liste de points, un point `p`, un entier `k` et une fonction distance `d` et qui renvoie la liste des `k` plus proches voisins de `p` déterminée à l'aide de l'algorithme cité.

Exercice n°3 Trier des points

On dispose d'un ensemble de points dans l'espace muni d'un repère orthonormé d'origine O . Un point possède des coordonnées (x, y, z) représentées par la liste [x, y, z]. L'objectif est de trier ces points en fonction de leur distance à un point donné P , de coordonnées (x_p, y_p, z_p), de la plus petite à la plus grande.

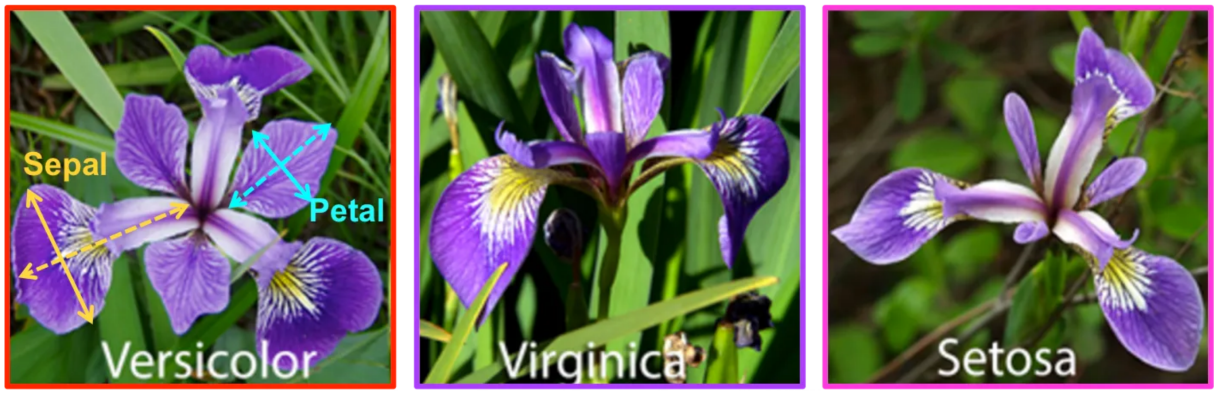
Q1. Écrire une fonction `distance` qui prend en paramètres deux listes représentant les coordonnées de deux points quelconques et renvoie le carré de la distance euclidienne entre ces deux points.

Q2. Écrire une fonction `compare` qui prend en paramètres trois listes `p`, `p1`, `p2` représentant dans l'ordre le point P et deux points quelconques P_1 et P_2 et qui renvoie -1 si P_1 est plus proche de P que P_2 ; 1 si P_2 est plus proche de P que P_1, et 0 si les deux points sont équidistants de P .

Q3. Écrire une fonction `tri_points` qui prend en paramètres une liste représentant un point `P` et une liste composée de listes de trois nombres représentant des points du plan et qui trie cette liste par ordre croissant des distances entre chaque point et `P`. Utiliser par exemple le tri par sélection. Comment obtenir un tri dans l'ordre inverse ?

II Problèmes

Exercice n°4 Les iris



On utilise les données disponibles de la bibliothèque `Scikit-Learn` sur les iris. Les caractéristiques numériques (longueur du sépale, largeur du sépale, longueur du pétale, largeur du pétale) des iris de la base de données sont contenues dans le tableau `iris.data` à deux dimensions. `iris.data[i]` est un vecteur de $\mathbb{R}^4$ qui contient les quatre caractéristiques de l'iris `i`.
`iris.target[i]` est un entier (0, 1 ou 2) qui renvoie à la variété de l'iris `i`. Cet entier est le rang de la liste `iris.target_names` des variétés.

```
from sklearn.datasets import load_iris
iris=load_iris()
>>> iris.target_names
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
>>> iris.feature_names # liste des caractéristiques prises en compte
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
>>> iris.data # caractéristiques des différents iris
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       ...
       ])
>>> iris.target # donne la variété de l'iris (plus exactement son rang dans
   la liste iris.target_names) dans le même ordre que iris.data
array([0, 0 ,...
       .... 2, 2 ])
```

L'objectif est de déterminer la variété d'un iris inconnu, connaissant les quatre caractéristiques (longueur du sépale, largeur du sépale, longueur du pétale, largeur du pétale.

Q1. Écrire la fonction `distance(x:list,y:list)->float` qui renvoie la distance au carré entre les deux vecteurs `x` et `y` de $\mathbb{R}^4$.

Q2. Expliquer le principe de l'algorithme des k-plus proches voisins.

Q3. Écrire la fonction `distances(C:array,V:array,X:array)->list` qui prend en arguments :

- `C` la liste de listes des caractéristiques des iris des données d'apprentissage (`C` est de la forme de `iris.data`),
- `V` la liste des variétés des données d'apprentissage (`V` est de la forme de `iris.target`)
- `X` la liste (de quatre éléments) des caractéristiques de l'iris dont on souhaite déterminer la variété,

et qui renvoie la liste de listes `dist` de deux éléments :

- le premier élément de `dist[i]` est la distance au carré entre `X` et `C[i]`,
- et le deuxième élément est la variété de l'iris `i`.

La liste renvoyée sera triée par ordre de distance croissante. Pour cela, on pourra utiliser `L.sort()` qui trie par ordre croissant `L` (par défaut, si `L` est une liste de listes, cela trie selon le premier élément de chaque sous liste).

Q4. Écrire la fonction `rang_max(L:list)->int` qui prend en argument une liste `L` de flottants et renvoie le rang du maximum des valeurs de `L`.

Q5. Écrire la fonction `kppvoisin(C:array,V:array,X:array,k):int` qui prend en arguments `C` la liste des listes des caractéristiques des iris des données d'apprentissage (`C` est de la forme de `iris.data`), `V` la liste des variétés des données d'apprentissage (`V` est de la forme de `iris.target`), `X` la liste (de quatre éléments) des caractéristiques de l'iris dont on souhaite déterminer la variété, et `k` un entier qui est le nombre de plus proches voisins considérés, et qui renvoie la variété de l'iris inconnu (plus exactement son rang dans la liste `iris.target_names`) en utilisant l'algorithme des k plus proches voisins.

Q6. Rappeler la définition de la matrice de confusion. Quelle est sa taille pour la situation qui nous intéresse ? Quelle information donnent les éléments sur la diagonale ? hors de la diagonale ?

Q7. On obtient la matrice suivante : $\begin{pmatrix} 20 & 0 & 0 \\ 0 & 12 & 7 \\ 0 & 2 & 9 \end{pmatrix}$. Commenter.

Les données numériques sont d'ordres de grandeur différents selon les caractéristiques. Par conséquent, la longueur des sépales a une plus grande importance dans le classement que la largeur des pétales. Pour éviter cela, il faut normaliser les données, soit les ramener dans l'intervalle $[0, 1]$.

Q8. Comment peut-on normaliser les données ?

Proposer une expression de la donnée numérique d'_i (par exemple la longueur des sépales) normalisée linéairement à partir de la donnée d_i, et des valeurs minimale et maximale de cette donnée (par exemple des longueurs des sépales).

Q9. Proposer une fonction `normaliser` qui prend en argument la liste des données, et renvoie une nouvelle liste de même nature et dans le même ordre qui contient les données normalisées.

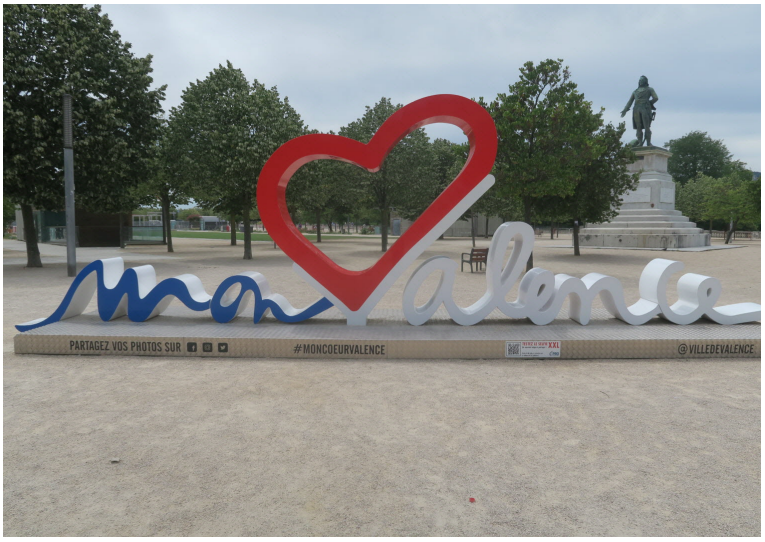
On rappelle que :

— pour récupérer toute la colonne de rang i dans un tableau `T` (`numpy`) à deux dimensions, on peut écrire simplement `T[:,i]`

— pour récupérer toute la ligne de rang i dans un tableau `T` (`numpy`) à deux dimensions, on peut écrire simplement `T[i,:]`

Exercice n°5 Compression d'image

On considère l'image ci-dessous (la mairie de Valence avait paré le célèbre « Mon cœur Valence » du drapeau français à l'occasion de la victoire à la coupe du Monde de football 2018 ...), de dimension 1067×1519 pixels.



Chaque pixel est représenté par un triplet (r, g, b) où r, g et b sont des entiers codés sur 8 bits (donc entre 0 et 255) représentants la quantité de rouge, vert et bleu du pixel. L'image est représentée en machine par un tableau `numpy` à trois dimensions : le pixel de la ligne i et de la colonne j est représenté par une liste de trois entiers $[r, g, b]$ qui est l'élément de rang j de la liste de rang i.

Q1. Écrire un script `python` qui calcule le nombre de couleurs différentes utilisées dans cette image.

On pourra créer un dictionnaire, dont les clés seront les 3-uplet caractérisant la couleur des pixels. Si la couleur n'a pas déjà été rencontrée (c'est-à-dire si le 3-uplet n'est pas dans le dictionnaire), il sera ajouté au dictionnaire (avec la valeur associée 1 par exemple).

Il restera à renvoyer la longueur du dictionnaire.

On obtient 68796 couleurs différentes. Notre objectif est de réduire ce nombre à seulement 16 couleurs. Pour ce faire, nous allons appliquer l'algorithme des k-moyennes pour regrouper les différents pixels en 16 classes, calculer la couleur moyenne de chacune de ces 16 classes, puis attribuer cette valeur moyenne à chacun des pixels de la classe correspondante.

La variable `img` est un tableau numpy à 3 dimensions : $1067 \times 1519 \times 3$ et représente la photo que l'on veut compresser.

Q2. Écrire une fonction `dist(p,q)` qui prend pour arguments deux pixels `p` et `q` (représentés par deux vecteurs dans $\mathbb{R}^3$) et renvoie la distance euclidienne entre ces deux vecteurs.

Q3. Écrire une fonction `initialise(img,k)` qui prend pour arguments une image `img`, un entier `k` et renvoie un tableau `numpy` de k cases, chacune d'elles contenant un pixel tiré au hasard dans l'image.

Rq : on commencera par créer une liste de k éléments que l'on remplira comme indiqué, puis on finira par convertir la liste en tableau `numpy` avec `np.array(liste)`.

Q4. Écrire une fonction `barycentre(img,s)` qui prend pour argument une image `img` et un ensemble `s` de coordonnées (x, y) et renvoie un pixel (c'est-à-dire un triplet $[r, g, b]$) égal au barycentre (en terme de couleurs) des pixels de l'image dont les coordonnées appartiennent à s.

Q5. Écrire une fonction `PlusProchePixel(p,mu)` qui prend pour argument un pixel p et une liste de pixels $[\mu_0, ..., \mu_{k-1}]$ et qui renvoie l'indice j qui minimise la distance $\|p - \mu_j\|$.

Q6. En déduire une fonction `kmoyennes(img, k)` qui prend pour arguments une image `img` et un entier `k` et qui renvoie un tableau `s` de longueur `k`, chacun de ses éléments étant un ensemble de coordonnées des pixels obtenu par l'algorithme des k-moyennes.

Une fois la partition obtenue, il reste à calculer la couleur moyenne de chacune des classes et attribuer cette couleur à chacun des pixels de la classe.

Q7. Rédiger une fonction `reduire(img, k)` qui prend pour argument une image et renvoie une nouvelle image dans laquelle seules `k` couleurs sont utilisées.