TP 07 - IA

Le but de ce TP est d'implémenter les algorithmes d'apprentissages automatiques vue en cours sur un exemple simple : la reconnaissance de caractères.

I Importations d'images

Dans tout ce TP, nous construirons des caractères (chiffres ou lettres) grâce à paint, un logiciel capable d'éditer des dessins sous format .png.

Pour ce faire, vous commencerez par créer un dossier, du nom de votre choix quelque part dans votre espace personnel, afin d'y sauvegarder l'intégralité des fichiers nécessaires au bon fonctionnement de ce TP. Ensuite, vous ouvrirez pyzo et créerez un nouveau document que vous enregistrerez dans le dossier dernièrement créé. L'entête de ce nouveau document .py devra commencer par :

```
import numpy as np
from PIL import Image
import random as rd
```

La fonction `Image` du module `PIL` permet d'importer des fichiers image de format .png grâce à la commande :

```
Image.open("nom_de_l_image.png")
```

Le module `numpy` permet de convertir cette importation sous la forme d'un `array` (type tableau du module numpy) de dimensions `largeur`×`hauteur`×3. Chacun des `largeur` éléments du tableau principal correspond à une ligne de pixels de l'image. Chacun des éléments d'un sous tableau correspond à un pixel particulier. Enfin, chaque pixel est enregistré sous la forme d'un tableau à trois éléments de type `uint8`, c'est à dire des entiers enregistré sur 8 bits (entre 0 et 255), correspondants respectivement à l'intensité de rouge puis de vert puis de bleu du pixel. (C'est pourquoi on parle d'image RGB, red-green-blue).

Lorsque vous importerez puis convertirez une image en tableau numpy, il sera judicieux de sauvegarder la valeur obtenue dans une variable. Aussi, votre fichier python devrait être parsemé de lignes ressemblant à :

```
nom_de_variable = np.array(Image.open("nom_de_l_image.png"))
```

Par exemple :

```
un_1 = np.array(Image.open("1_1.png"))
```

si l'image représentant votre premier un est appelé 1_1.png.

II Fonctions utilitaires

Question 1. Ecrire une fonction `noir_blanc(image)` prenant en argument une image sous la forme d'un tableau numpy de dimension 3 et renvoyant cette image convertit en une image en noir et blanc codée par une liste de listes dont le `j`$^{\text{ième}}$ élément de la `i`$^{\text{ième}}$ sous liste sera 0 si le pixel correspondant est blanc (toutes les intensités de couleurs sont à 255) et 1 sinon (on dira que c'est "noir").

Indications.

- La fonction `np.shape` permet de récupérer un tuple contenant les tailles des tableaux numpy. Ainsi `n,m,p = np.shape(image)` permet d'avoir dans `n` et `m` la hauteur et la largeur de notre image et `p` prendra la valeur 3.
- On pourra tricher un peu, compte tenu que les images que l'on fera sous paint seront uniquement en noir et blanc (sauf si vous voulez vous amusez avec de la couleur).

Question 2. Ecrire une fonction `premier_gauche(image)` prenant en argument une image au format noir et blanc (liste de listes) et récupérant l'indice de colonne du pixel noir le plus à gauche de notre image. Si l'image est entièrement blanche, cette fonction devra renvoyer l'indice de la dernière colonne.

Question 3. Ecrire une fonction `premier_droite(image)` prenant en argument une image au format noir et blanc (liste de listes) et récupérant l'indice de colonne du pixel noir le plus à droite de notre image. Si l'image est entièrement blanche, cette fonction devra renvoyer l'indice de la première colonne (c'est à dire 0).

Question 4. Ecrire une fonction `premier_haut(image)` prenant en argument une image au format noir et blanc (liste de listes) et récupérant l'indice de ligne du pixel noir le plus haut de notre image. Si l'image est entièrement blanche, cette fonction devra renvoyer l'indice de la dernière ligne.

Question 5. Ecrire une fonction `premier_bas(image)` prenant en argument une image au format noir et blanc (liste de listes) et récupérant l'indice de ligne du pixel noir le plus bas de notre image. Si l'image est entièrement blanche, cette fonction devra renvoyer l'indice de la première ligne (c'est à dire 0).

Question 6. Ecrire une fonction `zoom_car(image)` prenant en argument une image au format noir et blanc (liste de listes). Cette fonction devra récupérer les indices des lignes des pixels noirs les plus haut et plus bas de l'image dans des variables `h` et `b` ainsi que les indices des colonnes des pixels noirs les plus à gauche et à droite de l'image dans des variables `g` et `d`. Elle s'assurera alors, grâce à la commande `assert` que l'image n'est pas que blanche en testant si `h` est bien plus petit que `b` et si `g` est bien plus petit que `d`. Enfin, elle renverra une copie de la sous-image contenant le caractère, c'est à dire partant de l'indice de ligne `h` pour aller jusqu'à l'indice de ligne `b` et partant de l'indice de colonne `g` pour aller jusqu'à l'indice de colonne `d`.

Question 7. Ecrire une fonction `redimensionnage(image,largeur,hauteur)` prenant en argument une image en noir et blanc et la redimensionnant en une image ayant une résolution de `largeur`×`hauteur`. Pour calculer chaque pixel de la sortie, on considérera la couleur dominante (blanche ou noire) parmi les pixels de l'entrée dont l'intersection avec notre pixel de sortie considéré est non vide.

Indication. Si l'image en entrée est de résolution $m \times n$ alors n correspond au nombre de lignes formant l'image d'entrée et le haut de la $i^{\text{ième}}$ ligne de pixels du résultat se trouve dans la ligne de pixels d'indice $\left\lfloor \frac{i \times n}{\texttt{hauteur}} \right\rfloor$ de l'entrée.

III Apprentissage supervisé

III.A Construction de la base

Il est temps d'ouvrir paint afin de faire des dessins. Une fois le logiciel ouvert, afin d'éviter d'avoir de tout et n'importe quoi dans votre dossier, commencez par redimensionner la zone de dessin. Pour cela, cliquez sur la petite icône "redimensionner" (à côté de "sélectionner"), par exemple 100 pixels par 100 pixels. Assurez vous que la case "conserver les proportions" soit décochée.

Une fois la configuration initiale prête, dessinez au pinceau et en noir un chiffre puis enregistrez le dans le même dossier que le fichier python avec un nom approprié. Ensuite faite un nouveau document et refaite un chiffre etc....

Dans l'absolu, vous pouvez en faire le nombre que vous voulez. Par exemple, cinq versions de chaque chiffre appelées respectivement "0_1.png", "0_2.png",....., "9_5.png".

Une fois ceci fait, ouvrez chacune de ces images et enregistrez les sous un nom approprié avec python. Pour ce faire, utiliser l'instruction décrite en début de TP en exécutant le fichier comme un script. Pour ce faire, allez dans "exécuter" dans la barre des tâches (run en anglais), entre terminal et outils. Cette opération est nécessaire, elle permet d'indiquer à pyzo dans quel dossier il faut chercher les images.

Question 8. Une fois les images créées, enregistrez dans une variable `liste_classe` la liste `[0,1,2,3,4,5,6,7,8,9]` et dans une variable `base_brute` une liste de couples dont la seconde composante est une image importée précédemment et la première composante est sa classe.

Question 9. Ecrire une fonction `construire_base(pre_base,largeur,hauteur)` prenant en arguments une `pre_base` comme décrite à la question précédente et deux entiers `largeur` et `hauteur`. Cette fonction renverra une base sous la forme d'une liste de couples dont la première composante sera une classe d'image et la seconde une image de la `pre_base` à laquelle on a appliqué les fonctions `zoom_car` et `redimensionnage` avec les paramètres `largeur` et `hauteur` correspondants.

III.B Algorithme des k plus proches voisins

Question 10. Nous allons commencer par munir l'ensemble des images noir et blanc d'une distance (dite de Hamming). Ecrire une fonction `distance_hamming(image1,image2)` prenant en arguments deux images de même tailles (cette fonction pourra le vérifier grâce à l'instruction `assert`) et renvoyant le nombre de pixels différents entre les deux images.

Nous aurons ensuite besoin de trier les éléments de notre base par distance croissante au caractère que l'on souhaite tester. Il nous faut donc une fonction de tri.

Question 11. Ecrire une fonction `fusion(L1,L2)` prenant en arguments deux listes de couples supposées triées dans l'ordre croissant par rapport à leur seconde composante. La fonction devra renvoyer une liste contenant tout les éléments présent dans `L1` et `L2` triés dans l'ordre croissant par rapport à leur seconde composante.

Question 12. Ecrire une fonction `tri_fusion(L)` prenant en argument une liste de couples et renvoyant cette même liste triée par ordre croissant par rapport à la seconde composante des couples.

Une fois la base triée, il faut encore trouver quelle classe est dominante parmi les k plus proche voisins. Pour ce faire, nous allons nous doter de fonctions utilitaires qui peuvent paraître superflues pour la reconnaissances de chiffres mais qui vous permettraient d'adapter votre code facilement à la reconnaissance de lettres.

Question 13. Ecrire une fonction `incrementation(compteur,liste_classes,classe_elem)` prenant en arguments une liste de classes `liste_classes`, une liste d'entiers `compteur` de même taille que `liste_classes` et la classe d'un élément `classe_elem`. Cette fonction devra incrémenter de 1 la valeur de la case de la liste `compteur` d'indice correspondant à l'indice de la case de `liste_classes` contenant la valeur `classe_elem`.

Question 14. Ecrire une fonction `resultat(compteur,liste_classes)` prenant en arguments une liste de classes `liste_classes` et une liste d'entiers `compteur` de même taille que `liste_classes`. Cette fonction devra renvoyer l'élément de `listes_classes` présent dans la case d'indice correspondant à l'indice du maximum de `compteur`

Et, pour finir :

Question 15. Ecrire une fonction `k_plus_proche_voisin(liste_classes,base,elem,k)` prenant en arguments une liste de classes `liste_classes`, une base (ou une pré base) `base`, une image en noir et blanc `elem` et un entier `k`. Cette fonction devra renvoyer la classe supposée de l'image `elem` par l'algorithme des k plus proches voisins.

Vous pouvez ensuite tester cette fonction sur diverses exemples que vous aurez fabriqués avec notre logiciel de dessins. N'hésitez pas à tester à la fois avec la pré-base et avec la base construite par la fonction `construire_base`. Normalement les résultats ne devraient pas être les mêmes (à moins que vous n'ayez un tracé vraiment régulier).

IV Apprentissage non supervisé

IV.A Initialisation et test sur les partitions

On rappel que l'algorithme des k-moyennes consiste en se donner un ensemble que l'on partitionne une première fois en k classes arbitraires dont on calcul les barycentres avant de redistribuer chaque élément de l'ensemble dans la classe dont le barycentre respectif est le plus proche. Et on réitère l'opération jusqu'à ce que notre partition ne bouge plus.

Il nous faut donc choisir un moyen de représenter une partition d'un ensemble et un moyen de construire une partition initiale aléatoire en k classes non vides.

Pour cela, puisque notre ensemble à p éléments (notre ensemble d'images de lettres .png importées puis redimensionnées à l'aide des fonctions du précédent TP) sera enregistré dans une liste de p éléments, nous allons représenter une partition en k classes par une liste de k listes. La $i^{\text{ième}}$ sous-liste de la partition contiendra les indices des éléments de notre ensemble présent dans la $i^{\text{ième}}$ classe de notre partition.

Remarque. Avec cette représentation, on a pas besoin de l'ensemble que l'on veut partitionner pour créer une partition mais juste son cardinal.

Question 16. Ecrire une fonction python `initialisation(p,k)` prenant en arguments deux entiers représentants respectivement la taille `p` de l'ensemble à partitionner le et nombre de classe `k` de la partition. Après s'être assurés que l'ensemble contient plus d'éléments que le nombre de classes demandé, cette fonction devra renvoyer une partition aléatoire en k classes toutes non vides.

Pour cela, on vous autorise (en plus des opérations classiques) les fonctions `rd.shuffle(L)` et `rd.randrange(0,k)` du module `random`. La première de ces fonctions mélange les éléments de la liste donnée en paramètre et la seconde renvoie un entier entre 0 et $k-1$.

Afin de savoir quand notre algorithme soit s'arrêter, il est également nécessaire de savoir quand deux partitions d'un même ensemble sont égales.

Question 17. Ecrire une fonction python `egal_partition(part1,part2,k)` prenant en arguments deux partitions `part1` et `part2` ainsi qu'un entier `k`. Après s'être assurée que chaque partition comprend bien `k` classes, cette fonction devra renvoyer un booléen caractérisant l'égalité entre les deux partitions.

IV.B Calcul de barycentre

Nous possédons, du TP précédent, une fonction `distance` nous donnant la distance, entre deux images en noir et blanc, définie comme le nombre de pixels différents. Cette distance est connue sous le nom de distance de Hamming et doit être associée à une notion particulière de barycentre.

Question 18. Ecrire une fonction python `barycentre_hamming(ensemble,classe)` prenant en argument deux listes, la première représentant l'ensemble d'images sur lequel nous allons appliquer les k moyennes et la seconde étant une liste d'indices d'éléments de la première représentant une classe d'une partition de la liste `ensemble`. Cette fonction devra renvoyer une liste de listes de 0 ou de 1 représentant le barycentre de Hamming du sous ensemble de `ensemble` représenté par `classe`.

Chaque pixel de ce barycentre est noir si les pixels équivalents des images de la classe de la partition `classe` sont majoritairement noir et blanc sinon.

Question 19. Ecrire une fonction python `plus_proche_hamming(liste_bar,elem)` prenant en argument une liste `liste_bar` d'image en noir et blanc et une image `elem`. Cette fonction devra renvoyer l'indice de l'image de la liste `liste_bar` la plus proche de `elem` au sens de la distance de Hamming.

IV.C Algorithme des k-moyennes

Question 20. Ecrire une fonction python `k_moyennes(ensemble,k)` prenant en argument une liste `ensemble` d'image en noir et blanc et un entier `k`. Cette fonction devra appliquer l'algorithme des k-moyennes afin de renvoyer une liste de listes représentant une partition en k classes de l'ensemble `ensemble` .