TP 6 : Apprentissage non supervisé pour la reconnaissance de chiffres manuscrits

Exercice 1 : Lecture des données

On travaille dans ce TP sur les données de la MNIST (Modified National Institute of Standard and Technology) handwritten digit database.

Cette base de données contient des images de chiffres manuscrits étiquetés (de 0 à 9).

Voici un extrait des images présentes dans cette base :

L'objectif de ce TP est de regrouper les images de cette base de données en 10 classes (correspondant aux 10 chiffres) en utilisant des algorithmes d'apprentissage non supervisé.

Nous utiliserons ensuite le fait que les données soient en réalité déjà étiquetées pour vérifier l'efficacité des différentes méthodes.

> **1)** Placer les fichiers `chiffres.ml`, `train-images-idx3-ubyte` et `train-labels-idx1` dans un même dossier.

La variable `images` contient les images de la base de données (il s'agit d'un `int array array array`) et la variable `etiquettes` contient les étiquettes de ces images (il s'agit d'un `int array`).

> **2)** Combien y a-t-il d'images dans la base de données ?

> **3)** Quelles sont les dimensions des images ?

4) Afficher une des images de la base de données.

De quel chiffre s'agit-il ? Vérifier en affichant son étiquette.

Exercice 2 : Algorithme des k-moyennes

Dans cet exercice, on souhaite appliquer l'algorithme des k-moyennes. Les valeurs des pixels des barycentres seront arrondies à l'entier le plus proche.

5) Écrire une fonction `distance : int array array -> int array array -> float` qui calcule la distance euclidienne entre deux images. On considèrera une image comme un vecteur de $\mathbb{R}^{28\times 28} = \mathbb{R}^{784}$.

6) Écrire une fonction `echantillon : int -> 'a array -> 'a array` qui prend en argument un entier m et un tableau t de taille N et renvoie un tableau de taille m contenant des éléments distincts de t choisis aléatoirement et uniformément. On interdira toute modification de t et on supposera $N \gg m$.

7) Écrire une fonction `plus_proche : int array array array -> int array array -> int` qui prend en argument un tableau d'images t (sous forme matricielle), une image I (sous forme matricielle) et renvoie l'indice de l'élément de t qui est le plus proche de I.

8) Écrire une fonction `classes : image array -> int array array array -> int array` qui prend en argument un tableau d'images étiquetées de taille N et un tableau de barycentres de taille m et renvoie un tableau classe de taille N contenant des éléments entre 0 et $m-1$, tel que la classe de l'image d'indice i correspond à l'indice du plus proche barycentre.

9) Écrire une fonction `barycentres : int -> image array -> int array -> int array array array` qui prend en argument un entier m, un tableau d'images étiquetées de taille N et un tableau de classes de taille N et renvoie un tableau de barycentre de taille m, le barycentre d'indice j étant calculé à partir de toutes les images de classe j.

10) En déduire une fonction `k_moyennes : int -> int -> image array -> int array` qui applique l'algorithme des k-moyennes pour attribuer une classe à chaque image. La fonction prendra en argument le nombre de classes, le nombre maximal d'itérations et le tableau d'images.

On souhaite maintenant vérifier la pertinence de cette fonction sur nos données.

11) En se servant de `etiquettes` modifier le tableau obtenu grâce à `k_moyennes` pour que le numéro attribué à chaque classe corresponde à l'étiquette majoritaire parmi les éléments de la classe.

On veut que la classe numéro 3 corresponde, autant que possible, à des images du chiffre 3.

12) Déterminer une matrice de confusion pour la classification obtenue.

13) Déterminer le taux d'erreur global.

Exercice 3 : Optimisations

On souhaite réduire le temps de calcul du programme obtenu dans l'exercice précédent. Pour ce faire, on remarque qu'une image contient une majorité de pixel noirs. Lors d'un calcul de distance, beaucoup de temps est donc passé à calculer des termes $(0-0)^2$. Pour éviter cela, on peut déterminer, pour chaque image, la liste de ses pixels significatifs (contenant de l'information).

14) Écrire une fonction `liste_blancs : int array array -> (int * int) list` qui prend en argument une matrice correspondant à une image en niveau de gris (qu'on pourra supposer de dimensions 28×28) et renvoie une liste de couples d'indices (i, j) telle que la luminosité est supérieure à 0 sur le pixel de coordonnées (i, j) dans l'image.

Pour éviter de recalculer la liste des pixels significatifs, on créera des tableaux de listes correspondant aux listes de pixels significatifs des données d'entraînement et de test.

15) Quelle est la proportion de pixels significatifs moyenne sur les images des données d'entraînement ?

16) Écrire une fonction `delta2 : int array array -> int array array -> (int * int) list -> (int * int) list -> float` qui prend en argument deux images, ainsi que les deux listes contenant leurs pixels significatifs, et renvoie la distance entre les deux images, en ignorant les pixels non significatifs.

17) Réécrire les fonctions de l'exercice précédent en modifiant ce qui est nécessaire pour prendre en compte ces pixels significatifs.

18) Vérifier que le temps de calcul est effectivement raccourci avec cette nouvelle méthode.

19) On peut encore améliorer le temps de calcul en ignorant tous les pixels dont la luminosité est inférieure à une certaine valeur (par exemple 10).

Modifier la fonction `liste_blancs` pour prendre en paramètre cette valeur seuil.

20) À partir de quelle valeur a-t-on un impact sur le taux d'erreur global ?

21) (bonus) Utiliser des fils d'exécution pour réduire davantage le temps de calcul.

Exercice 4 : Classification hiérarchique ascendante

22) De la même manière que l'exercice précédent, implémenter la classification hiérarchique ascendante pour obtenir 10 classes.

Il doit être possible d'essayer facilement différentes dissimilarités.

23) Quelle dissimilarité donne le meilleur résultat ?

24) (bonus) Optimiser votre implémentation comme à l'exercice 3.