TP_{26} Algorithme kNN

L'objectif de ce TP est d'exploiter l'algorithme kNN afin de faire de la reconnaissance de chiffres manuscrits. Vous disposez d'une base de code pour ce TP.

1 Base du MNIST

La base de données du MNIST est un jeu de données contenant des chiffres manuscrits. Cette base fournit :

- un jeu de 60000 images, avec les chiffres associés, destinées à l'entraînement ;
- un jeu de 10000 images, avec les chiffres associés, destinées aux tests.

Les images sont des carrés de 28 pixels de côté. Ces données sont accessibles à l'adresse suivante : `http://yann.lecun.com/exdb/mnist/`. On y trouvera également une description précise du format des fichiers. Pour ce TP, vous avez accès à un module `Mnist`, dont l'interface est décrite dans le fichier `mnist.mli`. Ses fonctions permettent de lire des données contenues dans un fichier IDX. La lecture d'une image donne des données dans un tableau d'entiers à une dimension, contenant l'image ligne par ligne.

Q1. Compléter la fonction `print_image` du fichier `print_mnist.ml` afin d'écrire un programme qui affiche une représentation d'une image à l'écran, ainsi que son étiquette, à partir de son index dans la base du MNIST, passé en argument sur la ligne de commande. Dans les données, chaque pixel est un entier compris entre 0 (blanc) et 255 (noir). On peut choisir un jeu de caractères qui représentera chaque luminosité : le fichier `grayscale_70_levels.txt` vous suggère une liste de 70 caractères classiques pour représenter les tons allant du blanc au noir en ASCII art. Vous pouvez utiliser n'importe quelle autre représentation.

Pour tester, vous pouvez utiliser le `Makefile` fourni : la commande `make` lancera la compilation qui donnera un fichier exécutable `print_mnist`.

2 Boîte à outils

On va tenter de lire les données de manière paresseuse, c'est-à-dire sans nécessairement toujours charger les 60000 symboles en mémoire. Au lieu d'utiliser des listes, on utilise le type `Seq.t` du module `Seq`. Il est défini de la façon suivante :

```
type 'a node =
  | Nil
  | Cons of 'a * 'a t

and 'a t = unit -> 'a node
```

Un objet de type `Seq.t` est une fonction, qui prend un paramètre de type `unit`. L'élément suivant de la séquence n'est donc obtenu qu'en le demandant explicitement par un appel de fonction. Cette appel de fonction renvoie soit `Nil` pour indiquer que la séquence est terminée, soit `Cons (x, f)` où `x` est l'élément suivant et `f` est une fonction qui, quand on l'appellera, donnera la suite de la séquence.

Q2. Écrire une séquence, de type `Seq.t`, qui renvoie dans l'ordre les entiers 17 puis 42, puis s'arrête.

Écrire une séquence qui renvoie, à l'infini, une alternance de 17 et 42.

Q3. Écrire une séquence qui renvoie tous les entiers dans l'ordre, sans s'arrêter (en supposant que les entiers en machine ne sont pas bornés...).

Q4. Écrire une fonction `python_range` qui prend en paramètre deux entiers a et b et renvoie la séquence des entiers de a à $b-1$.

Vous avez également accès à un module `PrioQueue`, dont l'interface est décrite dans le fichier `prioQueue.mli`. Ce module implémente des files de priorité min non mutables à l'aide de tas. La création d'une file prend en paramètre une fonction de comparaison qui respecte la convention usuelle de la fonction `Stdlib.compare`. Il est donc possible d'utiliser la relation d'ordre de votre choix pour trier les données.

3 kNN sans prétraitement

Le but de cette partie est d'écrire un module `Knn` dans le fichier `knn.ml`, dont l'interface est décrite dans le fichier `knn.mli`. Les questions suivantes sont là pour vous guider dans l'écriture des fonctions attendues.

Q5. Écrire une fonction `count_item` qui compte le nombre d'occurrences d'un élément dans une liste.

Q6. Écrire une fonction `most_frequent` qui détermine, dans une liste non vide, un élément dont le nombre d'occurrences dans la liste est maximal.

Q7. Écrire une fonction `euclidean_dist : float array -> float arary -> float` qui calcule la distance euclidienne entre deux vecteurs de $\mathbb{R}^d$.

Q8. Écrire une fonction `mnist_seq` qui prend en paramètres un entier n et deux valeurs de type `Mnist.idx`, l'une pour le fichier d'images et l'autre pour le fichier de réponses, et qui construit une séquence de couples formées d'une image et de la réponse correspondante, contenant les couples associés aux n premières images.

Q9. Écrire une fonction `classify` qui prend en paramètres la séquence précédente, un entier k et une image du MNIST, et qui détermine la classe de cette image grâce à l'algorithme kNN.

Q10. Tester sur les images du jeu de tests. Déterminer en fonction de la valeur de k le taux d'erreur de l'algorithme de classification afin de déterminer la valeur de k la plus adaptée.

4 kNN avec arbres k-d

Le but de cette partie est de proposer une autre implémentation du module `Knn`, cette fois en exploitant les arbres k-d. Les questions ci-dessous peuvent vous guider.

Q11. Proposer un type pour représenter un arbre k-dimensionnel.

Q12. Écrire une fonction `median_coord` qui prend en paramètre une liste de points et une coordonnée et qui renvoie un triplet contenant le point médian selon cette coordonnée ainsi que les sous-listes des points respectivement strictement inférieurs et strictement supérieurs au point selon cette coordonnée.

Q13. Modifier la fonction `init` afin qu'elle construise l'arbre k-d, puis la fonction `classify` pour qu'elle exploite cet arbre.