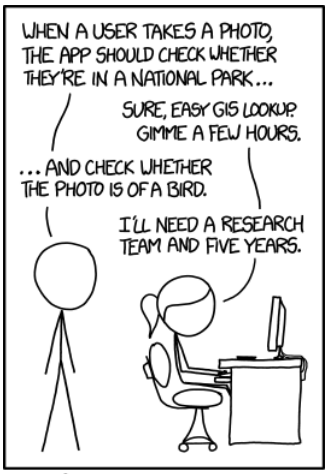
MPI* Info - TP11

Classification



R. Lahoute

O. Caffier

1 Lecture des données

1 Écrire une fonction `lire_lignes_fichier : in_channel -> string list` prenant en entrée un flux d'entrée et renvoyant une liste de chaînes de caractères dont le i-ème élément est la chaîne correspondant à la i-ème ligne sur le flux entrant.

Corrigé :

```
let lire_lignes_fichier (c : in_channel) : string list =
  let l = ref [] in
  try
    while true do
      l := input_line c :: !l
    done;
    (* On ne sortira pas de la boucle sans erreurs, mais il faut tout de meme renvoyer quelque chose pour
        eviter une erreur de typage *)
    !l
  with End_of_file -> List.rev !l
```

2 Écrire une fonction `parser_ligne_image : string -> int array` prenant en entrée une chaîne de caractères correspondant à la description d'une image et renvoyant un tableau d'entiers correspondant aux valeurs des pixels de cette image.

Corrigé :

```
let parser_ligne_image (s : string) : image =
  Array.of_list (List.map (fun x -> int_of_string x) (String.split_on_char ',' s))
```

3 Écrire une fonction `lire_images : string -> image array` prenant en entrée un nom de fichier contenant des images au format décrit par l'énoncé et renvoyant un tableau contenant en case `i` la représentation en Ocaml de la `i`-ème image.

Corrigé :

```
let lire_images (s : string) : image array =
  let c = open_in s in
  let l = lire_lignes_fichier c in
  close_in c;
  let rec aux l i t = (*l : liste des lignes, t : tableau renvoye, i : l'indice ou l'image est placee*)
    match l with
    | [] -> ()
    | x::xs ->
      begin
        t.(i) <- parser_ligne_image x;
        aux xs (i+1) t
      end
  in
  let t = Array.make (List.length l) [||]  in
  aux l 0 t;
  t
```

4 Écrire de même une fonction `lire_etiquettes : string -> array` prenant en entrée un nom de fichier contenant des étiquettes et renvoyant un tableau contenant la `i`-ème étiquette en case `i`.

Corrigé :

```
let lire_etiquettes (s : string) : int array =
  let l = ref [] in
  let c = open_in s in
  try
    while true do
      l := int_of_string (input_line c) :: !l
    done;
    (*On ne sortira pas de la boucle sans erreurs, mais il faut tout de meme renvoyer quelque chose pour
        eviter une erreur de typage*)
    Array.of_list (List.rev !l)
  with End_of_file ->
    begin
      close_in c;
      Array.of_list (List.rev !l)
    end
```

5 Écrire une fonction `jeu_donnees : image array -> int array -> (image*int) array` prenant en entrée un tableau d'images et le tableau d'étiquettes correspondantes et renvoyant le tableau associant chaque image à son étiquette.

Corrigé :

```
let jeu_de_donnees (t_im : image array) (t_et : int array) : (image*int) array =
  let n = Array.length t_im in
  let t_do = Array.make n ([||],0) in
  for i = 0 to n-1 do
    t_do.(i) <- (t_im.(i),t_et.(i));
  done;
  t_do
```

6 Écrire une fonction `affiche_image : image -> unit` prenant en entrée un tableau d'entiers représentant une image et l'affichant. Pour chaque pixel, il faudra décider quel symbole dessiner en fonction de sa valeur. On rappelle que les images sont de taille 28×28 pixels.

```
let affiche_image (im : image) : unit =
  let cpt = ref 0 in
  for i = 0 to Array.length im -1 do
    if !cpt >= 28 then
      begin cpt := 0; print_newline() end;
    incr cpt;
    if im.(i) <= 50 then print_char '.'
    else if im.(i) <= 101 then print_char '-'
    else if im.(i) <= 152 then print_char '/'
    else if im.(i) <= 203 then print_char '$'
    else print_char '#'
  done
```

7 À l'aide des fonctions précédentes, afficher quelques uns des exemples (image,étiquette) du jeu d'apprentissage fourni.

Corrigé :

```
let _ =
  let t_im = lire_images "x_train.csv" in
  let t_et = lire_etiquettes "y_train.csv" in
  let donnees = jeu_de_donnees t_im t_et in
  for i = 0 to 9 do
    let im,v = donnees.(i) in
    print_int v;
    print_newline();
    affiche_image im;
    print_newline()
  done;
```

2 Méthode des k plus proches voisins

```
type modele = {distance : image -> image -> float ;
  k : int;
  donnees : (image * int) array}
```

8 Écrire une fonction `norme_1 : image -> image -> float` prenant en entrée deux images et renvoyant la distance qui les sépare selon la norme 1 sous forme d'un flottant.
Corrigé :

```
let norme_1 (im1 : image) (im2 : image) : float =
  let dist = ref 0. in
  for i = 0 to Array.length im1 -1 do
    dist := !dist +. (abs_float (float_of_int(im1.(i)) -. float_of_int(im2.(i))))
  done;
  !dist
```

9 Écrire une fonction `nb_elements : int list -> int array` prenant une liste l d'éléments de $[\![0;9]\!]$ et renvoyant un tableau de 10 cases contenant en case i le nombre d'occurences de i dans l.
Corrigé :

```
let nb_elements (l : int list) : int array =
  let t = Array.make 10 0 in
  let aux l =
    match l with
    | [] -> ()
    | x::xs -> t.(x) <- t.(x) + 1
  in
  aux l;
  t
```

10 Écrire une fonction `classe_plus_frequente : int list -> int` renvoyant l'un des entiers le plus fréquent dans une liste l d'éléments de $[\![0;9]\!]$.
Corrigé :

```
let classe_plus_frequente (l : int list) : int =
  let t = nb_elements l in
  let maxi = ref t.(0) in
  let classe_max = ref 0 in
  for i = 1 to Array.length t -1 do
    if t.(i) > !maxi then
      begin
        maxi := t.(i);
        classe_max := i;
      end
  done;
  !classe_max
```

11 Écrire une fonction `constuire_modele : (image -> image -> float) -> int -> (image*int) array -> modele` prenant en entrée une distance, un entier k et un jeu d'entraînement et qui renvoie le modèle correspondant selon la méthode des k plus proches voisins.
Corrigé :

```
let construire_modele dist k donnees : modele =
  {distance = dist; k = k; donnees = donnees;}
```

12 Écrire enfin une fonction `determnier_classe : modele -> image -> int` déterminant la classe de l'image en entrée selon le modèle qu'est le premier argument.

Corrigé :

```
let determiner_classe (m : modele) (im : image) : int =
  let tab = Array.copy m.donnees in

  (*Pour utiliser Array.sort (ou stable_sort), il faut pouvoir comparer les elts (ici images) entre elles,
      ce que fait fct_comp en utilisant la distance a im, l'image que l'on veut classer*)
  let fct_comp a b =
    let d1 = m.distance (fst a) im in
    let d2 = m.distance (fst b) im in
    if d1 > d2 then 1 else if d1 = d2 then 0 else -1
  in

  (*On trie les donnees du modele par ordre croissant de distance a im*)
  Array.stable_sort fct_comp tab;
  let l = ref [] in

  (*on recupere les k classes les plus proches*)
  for i = 0 to m.k -1 do
    l := (snd tab.(i)) :: !l
  done;

  (*et on renvoie la plus frequente*)
  classe_plus_frequente !l
```

Remarque : il serait plus efficace de travailler avec une file de priorité max pour récupérer les k classes les plus proches, afin de ne pas passer par le tri et ainsi gagner en complexité temporelle.

13 Construire le modèle pour lequel $k = 2$ et la distance est celle donnée par la norme 1. Déterminer la classe prédite sur quelques exemples de l'ensemble de tests à l'aide de la fonction précédente. La classe prédite est-elle toujours celle attendue?

Corrigé :

```
let _ =
  let im_test = lire_images "x_test.csv" in
  let et_test = lire_etiquettes "y_test.csv" in
  let jeu_test = jeu_de_donnees im_test et_test in

  let im_train = lire_images "x_train.csv" in
  let et_train = lire_etiquettes "y_train.csv" in

  let donnees = jeu_de_donnees im_train et_train in

  let k = 2 in
  let m = construire_modele norme_1 k donnees in

  for i = 0 to 9 do
    let im,v = jeu_test.(i) in
    let classe = determiner_classe m im in
    affiche_image im;
    print_newline();

    print_string "Classe attendue : ";
    print_int v;
    print_newline ();

    print_string "Classe trouvee : ";
    print_int classe;
    print_newline ()
  done
```

3 Évaluation de modèles

14 Écrire une fonction `pourcentage_erreur : modele -> (image*int) array -> float` prenant en entrée un modèle m et un jeu de test E et déterminant le pourcentage d'éléments de E qui ont été mal classifiés par m.

Corrigé :

```
let pourcentage_erreur (m : modele) (jeu : (image*int) array) : float =
  let n = Array.length jeu in
  let mal_classifie = ref 0 in
  for i = 0 to n-1 do
    let classe = determiner_classe m (fst jeu.(i)) in
    if classe <> snd jeu.(i) then
      incr mal_classifie
  done;
  ((float_of_int !mal_classifie) /. (float_of_int n)) *. 100.
```

15 Écrire une fonction `matrice_confusion : modele -> (image*int) array -> int array array` calculant la matrice de confusion d'un modèle sur un jeu de test.

Corrigé :

```
let matrice_confusion (m : modele) (jeu : (image*int) array) : int array array =
  let n = Array.length jeu in
  let mat_conf = Array.make_matrix 10 10 0 in
  for k = 0 to n-1 do
    let classe = determiner_classe m (fst jeu.(k)) in
    mat_conf.(snd jeu.(k)).(classe) <- mat_conf.(snd jeu.(k)).(classe) + 1
  done;
  mat_conf
```

16 Tester la fonction précédente sur le modèle construit à la question 13 et commenter les résultats obtenus.

```
let _ =
  let im_test = lire_images "x_test.csv" in
  let et_test = lire_etiquettes "y_test.csv" in
  let jeu_test = jeu_de_donnees im_test et_test in
  let sous_jeu_test = Array.sub jeu_test 0 20  in (*jeu_test est trop grand, on le reduit pour la matrice de
       confusion afin d'avoir des calculs en temps raisonnable*)

  let im_train = lire_images "x_train.csv" in
  let et_train = lire_etiquettes "y_train.csv" in
  let donnees = jeu_de_donnees im_train et_train in
  let k = 2 in
  let m = construire_modele norme_1 k donnees in

  let mat_conf = matrice_confusion m sous_jeu_test in
  for i = 0 to 9 do
    for j = 0 to 9 do
      print_int mat_conf.(i).(j);
      print_string " ";
    done;
    print_newline ();
  done
```

⇒ La matrice obtenue n'est pas diagonale : La classification n'est pas parfaite. (Le calcul a été long...)

17 Calculer le pourcentage d'erreur de différents modèles construits avec la méthode des k plus proches voisins en faisant varier la distance utilisée (ou pourra essayer avec la distance euclidienne par exemple) ou le nombre k de voisins considérés pour déterminer la classe d'une image. On pourra examiner les images pour lesquelles des erreurs ont été produites et les afficher pour voir si une erreur était prévisible.

Corrigé :

```
let dist_eucl im1 im2 =
  let dist = ref 0. in
  for i = 0 to Array.length im1 -1 do
    dist := !dist +. (im1.(i) -. im2.(i))**2
  done;
  sqrt !dist
```

⇒ Le calcul avec la distance euclidienne et différents k est laissée à l'appréciation du lecteur.