Dimitri Charneux
Rémi Kroll

Rendu TP: Reconnaissance de caractères par codage de Freeman

1 Introduction

Dans le cadre de ce TP, nous avons chercher à reconnaître des chiffres à partir de plusieurs données d'apprentissage. Pour ce faire, nous allons calculer le codage de Freeman de chaque données (apprentissage et test) puis comparer ces codage au codage test grâce à la distance de Levenshtein. Enfin, nous avons utilisé un classifieur k-nn pour savoir dans quelle classe se situe notre donnée.

2 Préparation des données

Les données utilisées dans le TP vont être issu d'une version étendu de la base de données USPS. Elle contient des chiffres manuscrits issus de documents des services postaux américains. Chaque image correspond à une classe de 1100 images d'un meme chiffre (les chiffres vont de 0 à 9). Il y a donc 10 classes. Les images sont en niveaux de gris et ont des dimensions de 16 x 16 pixels.

Tout d'abord, nous avons du lire les 1100 x 10 images de la base de donnée, à l'aide de la méthode *readUSPSData()* (méthode fourni dans le fichier readUSPS.R). Pour pouvoir calculer le code de Freeman de chaque image, il faut toutes les binariser: nous avons utilisé la méthode *Threshold* du logiciel *ImageJ*. Par exemple pour la figure 1 montrant les images de la classe 0, nous avons la figure 2 qui correspond à son image binaire (voir annexe).

Enfin, nous pouvons maintenant appliquer le code de *Freeman* pour chaque image (le code nous a été fourni). Le code de *Freeman* nous permet de calculer la distance de Levenshtein entre chaque image.

3 Distance de Levenshtein

Nous pouvons calculer maintenant créer une fonction en R calculant la distance de Levenshtein entre deux codages de Freeman donnés selon la méthode vu en cours. Nous avons procédé de cette maniere :

```
levenshtein <- function( x, y ) {
  m = length(x)
  n = length(y)
  mat <- matrix(0 , nrow=m+1, ncol=n+1)
  for( i in 1:(m+1)){
    mat[i,1] = i-1
  }
  for( i in 1:(n+1)){
    mat[1,i] = i-1
  }
  for(i in 2:(m+1)){
    for(j in 2:(n+1)){
      tmp1 = mat[i-1,j]+1
      tmp2 = mat[i, j-1]+1
      tmp3 = mat[i-1,j-1]+1
      if(x[i-1] == y[j-1]){
          tmp3 = tmp3-1
      }
      mat[i ,j] = min(tmp1,tmp2,tmp3)
    }
  }
  return (mat[m+1,n+1]);
}
```

Pour calculer la distance entre deux mot, l'algorithme va utilisé une matrice de taille (|mot 1|, |mot 2|). La première ligne et la première colonne vont être initialisé avec leur numéro de ligne (resp colonne). Ensuite, pour remplir le reste des cases, l'algorithme va prendre le nombre minimum entre les trois nombres suivants : la case à gauche +1, la case en haut +1 et si les lettres coreespondant à la case sont égales, la case en haut à gauche, sinon, la case en haut à gauche +1. La distance de Levenshtein est la case situé en bas à gauche.

Voici un exemple pour illustrer le calcul de la distance de Levenshtein sur deux mots

		C	H	I	E	N	S
	0	1	2	3	4	5	6
N	**1**	1	2	3	4	4	5
I	**2**	2	2	2	3	4	5
C	3	**2**	3	3	3	4	5
H	4	3	**2**	**3**	4	4	5
E	5	4	3	3	**3**	**4**	**5**

:

Maintenant que nous avons vu comment est effectuer le calcul de Levenshtein, nous allons voir comment classifier une donnée par rapport aux données d'apprentissage.

4 Classifieur *k*-NN

Maintenant que nous savons calculer la distance de Levenshtein entre deux code de Freeman, nous pouvons classifier une donnée de test en fonction des données d'apprentissage du programme. Pour ce faire, nous avons utilisé un classifieur k-NN. Ce classifieur va calculer la distance de Levenshtein entre la donnée test et chacune des données d'apprentissage. Elle va ensuite prendre les k plus petites distances et regarder la classe la plus représentée dans ces k données. La donnée test sera mise dans cette classe.

Voici une représentation en pseudo-code de cette algorithme.

```
test la donnée à classifier
m = nombre de classe
n = nombre d'élément par classe
tab[m, n] un tableau a deux dimensions contenant les code de freeman des images
d'apprentissages
tabDistance[m, n] un tableau a deux dimensions contenant les distances entre la
donnée test et les données d'apprentissages

pour i allant de 1 à m
       pour j allant de 1 à n
              tabDistance[i, j] = levenshtein(tab[i,j], test)
       fin pour
fin pour

 tabMinDist[m] = tableaux du nombres d'élément de la classe m proche de test,
 initialisé à 0
pour i allant de 1 à k
       tmp = which.min(tabDistance)
       tabDistance[tmp] = un nombre très grand
```

```
       tabMinDist[tmp[0]] ++
fin pour
return which.max(tabMinDist)
```

5 Évaluation

Après avoir implémenté la fonction classifieur de type k-NN, il faut maintenant pouvoir effectuer un test d'une image en la comparant à des données tests. Pour cela, nous avons pris sept images aléatoires par classe et trois images de test par classe (pour des raisons de temps d'exécution, nous utilisons qu'une fraction des données initiales). La fonction qui nous permet d'effectuer cette méthode est la suivante :

```
list = list()
nbdonne_apprentissage = 7
nbdonne_test = 3
listdonne = array(0, dim=10)
cpt =1
for(i in 1:10){
  for(j in 1:nbdonne_apprentissage){
    x1 <- runif(1, 1, 1100)
    while( x1 %in% listdonne ){
      x1 <- runif(1, 1, 1100)
    }
    listdonne[j] = x1
    list[[cpt]] = tabFreeman[[(i-1)* 1100 + x1]]
    cpt = cpt +1
  }
  listdonne = array(0, dim= 10)
}
```

Cette méthode prend "nbdonne_apprentissage" par classe et les range dans une liste. Cette liste va ensuite être utilisé pour tester avec la méthode k-NN la classe dans laquelle va être mise une donnée.

Pour tester si ce programme est efficace, on va exécuter la méthode k-NN sur plusieurs données choisit aléatoirement dans notre et vérifier si la donnée a été rangée dans la bonne classe.

6 Conclusion

Nous avons pu voir comment classer une donnée de test avec des données d'apprentissage grâce à la méthode de *Freeman* et l'algorithme k-NN. Cependant, nous obtenons pas un résultat favorable. Nous pensons que les données utilisés pour raccourcir le temps d'exécution nous empêche d'obtenir un résultat correct.

Remarque :

La longueur du TP est correcte mais nous avons eu des difficultés à partir de la partie cinq, évaluation.

Annexe

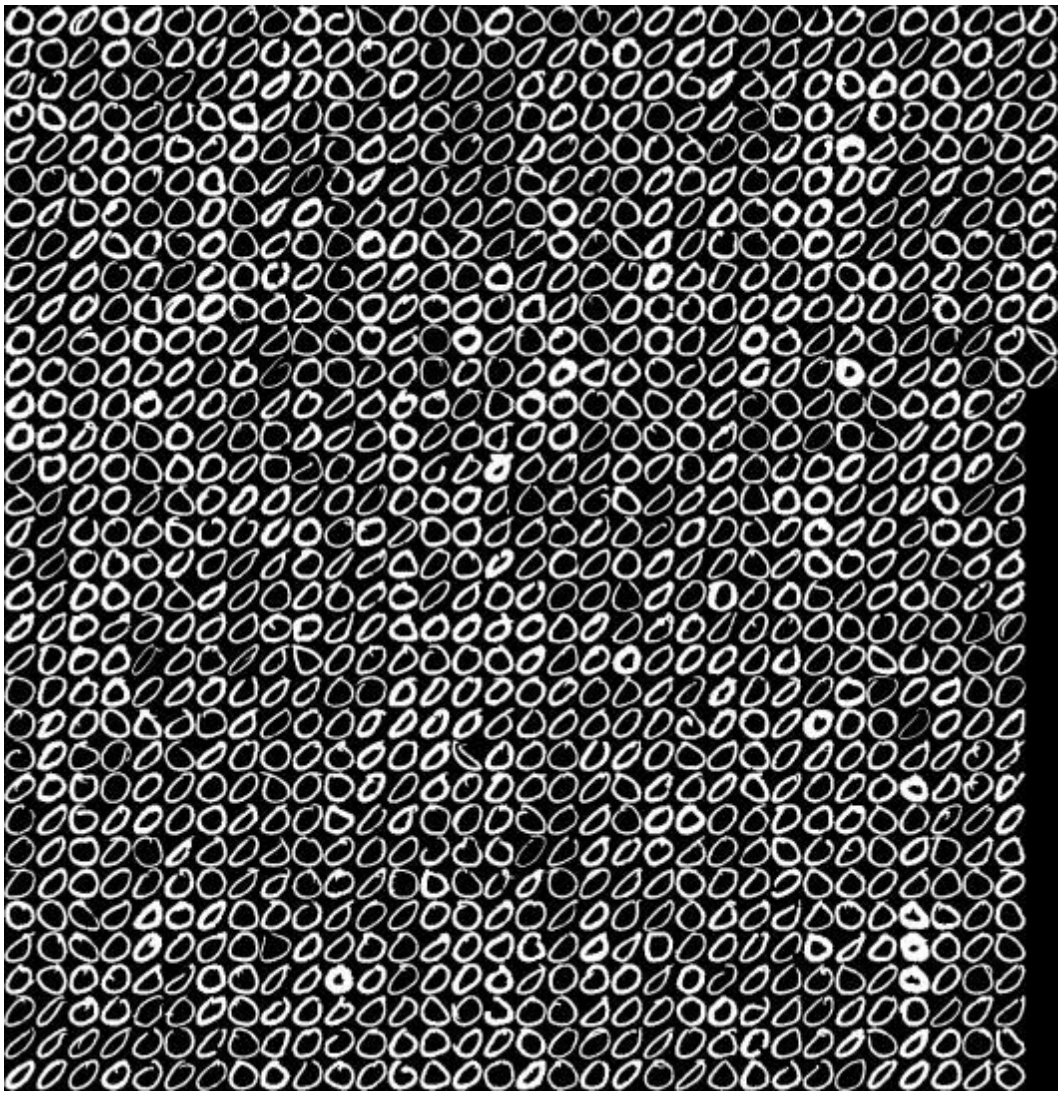

Figure 1 : *Images de la classe 0 des données USPS.*

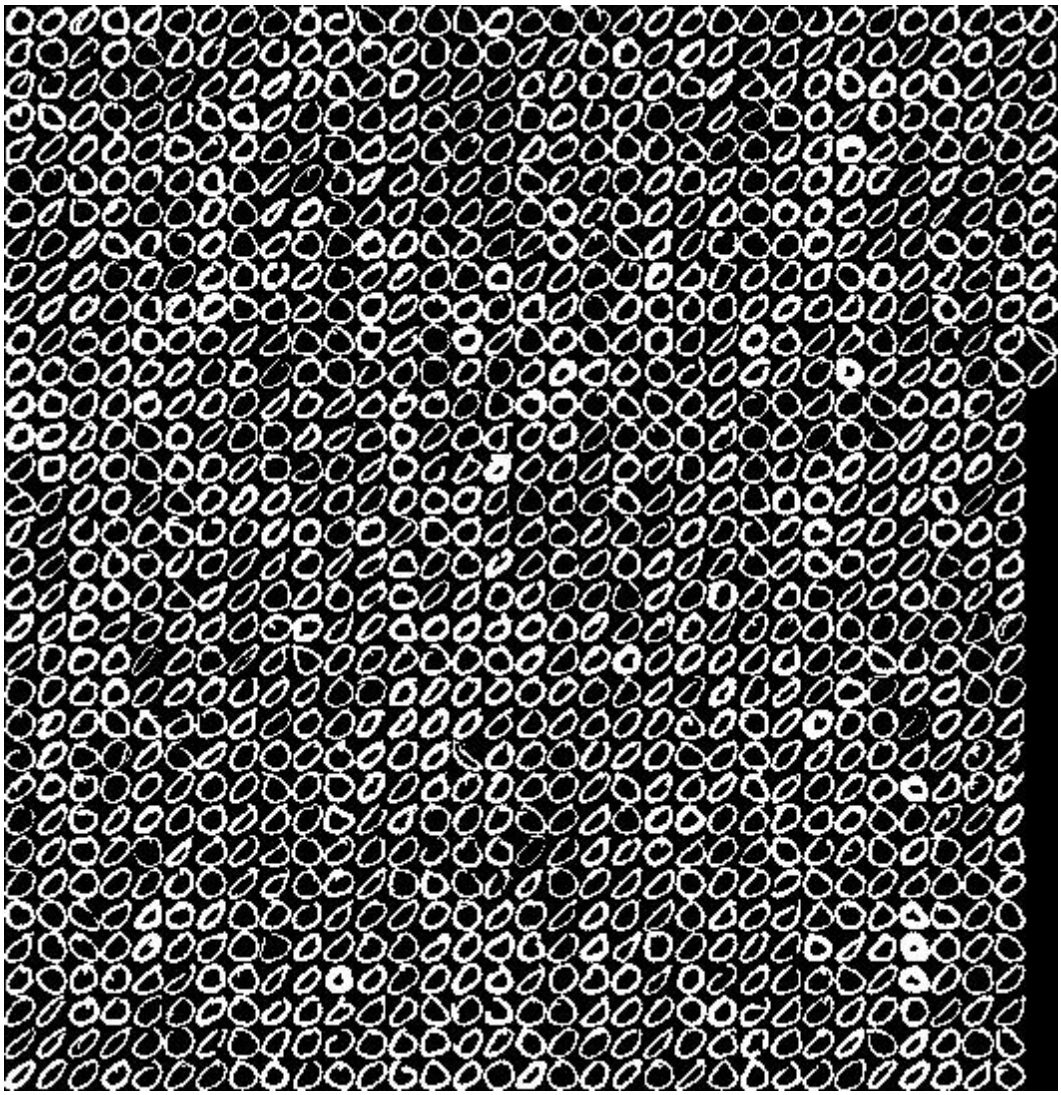

Figure 2: *Images de la classe 0 des données USPS binarisée*