Compte rendu TP10 RDF – Arbre de décision et reconnaissance de visages

Introduction

Nous avons vu dans le TP précédent l'utilisation des arbres de décision sur le problème du pendu. Lors de ces travaux pratiques, nous allons illustrer l'utilisation d'arbres de décision dans la reconnaissance de visages dans une base de données de 40 individus différents, chacun possédant un échantillon de 10 visages.

Préparation des données

La taille de chaque visage est de 33 pixels sur la largeur et 40 pixels en hauteur.

Remplissage de la matrice 3D avec les données des visages :

```
fillStack <- function (data, d1, d2, d3)
{
	data <- t(imageData(data));
	stackedFaces <- array(rep (0, d1*d2*d3), c(d1, d2, d3));

	for (i in 0:19)
	{
		for (j in 0:19)
		{
		stackedFaces[,, (i*20 + j + 1)] = data[(1+i*d1) : ((i+1)*d1), (1+j*d2) : ((j+1)*d2)];
		}
	}
	print ("End of filling");
	return (stackedFaces);
}
```

Cette fonction retourne une matrice de taille 40*33*400 qui contient les visages empilés.

Ensuite nous avons séparés les données test et apprentissage pour faciliter la construction de l'arbre de décision.

Méthodologie

L'arbre de décision sera construit en fonction d'un des pixels de l'image. La démarche sera la suivante : pour chaque pixel, à une position (x, y) donnée, nous calculerons sa variation entropie dans toutes les images I. Ensuite, on choisit parmi toutes les valeurs d'entropie la valeur la plus grande, puis nous utilisons ce pixel comme variable de décision pour construire notre arbre.

Une fois ce pixel choisi, on construit deux sous-listes : une avec les visages possédant ce pixel et l'autre avec les visages ne le possédant pas. Finalement, on vérifie si l'image pour laquelle nous cherchons la classe possède ou non ce pixel, afin de descendre sur l'une ou l'autre des branches de l'arbre.

La condition d'arrêt sera alors la taille de l'ensemble courant : on recommencera chacune de ses étapes dans l'ordre jusqu'à ce que l'ensemble courant ne possède plus qu'une seule classe, qui sera alors la classe que nous cherchons.

Différences avec le jeu du pendu

Dans le jeu du pendu on calculait l'entropie en parcourant chaque lettre de l'alphabet puis on choisissait la lettre possédant la meilleure entropie. En enfin on séparait l'ensemble des noms en 2 sous-ensembles :

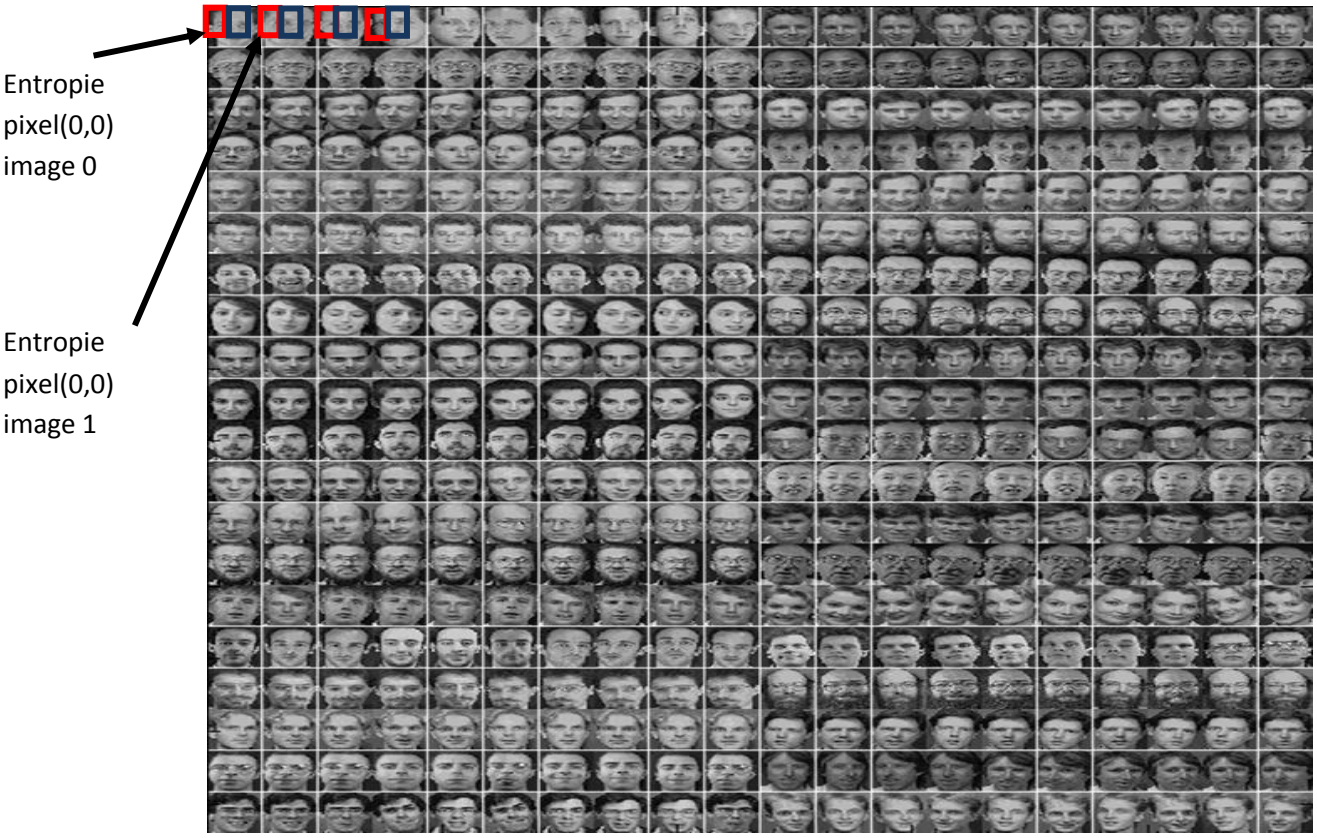
Noms qui contiennent la lettre | Noms qui ne contiennent pas la lettre

```
h <- rep (0, 26)
for (i in 1:26)
{
        h[i] = length(mat[i,][mat[i,]==1]) / length(liste_noms)
}

h <- -(log(h^h)) - (log ((1-h)^(1-h)))

lettre <- which.max(h)

list(let = lettre, avec = liste_noms[mat[lettre, liste_noms]==1], sans = liste_noms[mat[lettre, liste_noms]!=1])
```

Dans le cas des images avec les visages, il faut parcourir chaque pixel de l'image et calculer l'entropie de ce pixel pour toutes les images sélectionnées.

On va ensuite prendre la meilleure entropie parmi toutes celles calculées qui correspondent à l'entropie du pixel (0,0) de chaque image et répéter cela pour chaque pixel.

Illustration :



Réduction d'entropie

Comme cité précédemment, on va parcourir chaque pixel de l'image et de regarder dans quelles images le pixel s'y trouve. On va donc pour chaque pixel obtenir 2 sous-ensembles :

- Sous ensemble des images contenant ce pixel
- Sous ensembles contenant les images qui ne possèdent pas ce pixel

Mise en œuvre

Bien que la méthodologie ait été acquise, nous n'avons pas réussi à mettre en application ces principes. L'entropie calculée est somme toute bonne, mais il reste des soucis lors de la déduction des séparations et de la détection des classes. Nous n'avons pu déterminer d'où venait le souci.

Conclusion

Ces travaux pratiques bien qu'incomplets nous ont permis d'apprendre les méthodes permettant de séparer et reconnaître des formes via l'utilisation de l'entropie d'un pixel. Ils nous ont permis de comprendre les mécanismes derrière le calcul d'entropie et son utilité lors de la création d'un arbre de décision, permettant une détection automatique de formes telles que des visages.

Annexe – Code source

```
library("EBImage")
source("rdfSegmentation.R")


allFacesName="allFaces.png";
allFaces=rdfReadGreyImage(allFacesName);

# création de stackedFaces :
# Empilement des visages 40x33 dans une matrice stackedFaces 40x33x400
# Attention, exécution possiblement longue ~30 sec, pas de panique...)
# numLignes, numColonnes, numFaces


# Remplit une 'matrice 3D' (en vérité, un array) avec les
# valeurs contenues dans le data. C'est une fonction qui
# s'adresse tout particuliÃ¨rement au TP10 - 11 de RDF :)
```

```
fillStack <- function (data, d1, d2, d3)
{
	data <- t(imageData(data));
	stackedFaces <- array(rep (0, d1*d2*d3), c(d1, d2, d3));

	for (i in 0:19)
	{
		for (j in 0:19)
		{
	stackedFaces[,, (i*20 + j + 1)] = data[(1+i*d1) : ((i+1)*d1), (1+j*d2) : ((j+1)*d2)];
	}
	}

	print ("End of filling");
	return (stackedFaces);
}


fill <- function (data, filled, f, indice)
{
	for (i in 1:40)
	{
		for (j in 1:33)
		{
			filled [i,j,indice] <- data[i,j,f]
		}
	}

	return (filled)
}


separateData <- function (data)
{
	test <- array(rep (0, 40*33*400), c(40, 33, 400))
	app <- array(rep (0, 40*33*400), c(40, 33, 400))
	i <- 1
	j <- 1

	for (f in 1:400)
	{
		x <- sample (1:2, 1)
		if (x == 1)
		{
			test <- fill(data, test, f, i)
			i <- i+1
		}
		else
		{
			app <- fill(data, app, f, j)
```

```
                        j <- j+1
                }
        }

        print ("End of separation")
        display(test)
        display(app)
        return (list(app = app, test = test))
}

calculEntropie <- function(serie)
{
        entropie = matrix(rep(0,40*33),nrow=33,ncol=40);
        res = 0;
        for (i in 1:33) {
                for (j in 1:40) {
                        for (k in 1:400) {
                                res = res +serie[i,j,k];
                        }
                }
                m = res /400;
                entropie[i,j] <- -log2((res)^(res)) - log2((1-res)^(1-res));
        }
        entropie;
}



entropie <- function (stacked)  {
   entrop = zeros(40,33);
   res = 0;
   for (j in 1:40) {
      for (i in 1:33) {
         for (k in 1:400) {
            res = res  + stacked(jj,ii,kk);
         }
         res = res/400;
         entrop(j,i) = -log2(res^res) - log2((1-res)^(1-res));
      }
   }
}

faceStack <- fillStack(allFaces, 40, 33, 400);
list <- separateData (faceStack);
```