Reconnaissance des formes

TP9 : Arbres de décision

Par

Douaille Erwan & Francois Remy

Université Lille1
Sciences et Technologies

Introduction

Le but de ce TP est de nous faire comprendre comment déterminer au mieux les sous ensembles à éliminer/conserver en fonction d'un indice nous permettant d'optimiser au minimum le nombre de proposition que nous devrons faire pour trouver un élément parmis une liste. Concrètement avec le jeu du pendu, nous ferons une première analyse des mots dans laquelle nous choisirons quelle lettre proposer pour trouver le plus rapidement (donc avec un nombre de proposition minimal) le mot sélectionné par l'adversaire.

Question de bon sens

La valeur N vaut 16. Pour chaque essais on a 2 réponses possibles si ce n'est pas la bonne valeur (Supérieur ou Inférieur), de plus A indique qu'il faut exactement 4 propositions donc 2^4=16.

Quand A dit qu'il faut exactement 4 propositions, celà signifie que (si l'on raisonne par dichotomie) la valeur n'est pas un nombre pair (hormis 16), si le nombre choisi est 8 par B, ce qui permet d'éliminer ces valeurs et de ne choisir que des valeurs impairs ce qui permettrai de trouver plus rapidement la bonne valeur.

Variante du jeu du pendu

Question 2

La relation qui éxiste entre p et n est : $n = 2^p$

Question 3

La fonction :

```
alphabetIndex <- function(x) { strtoi(charToRaw(x),16L)-96 }
```

Permet d'indiquer la place dans l'alphabet qu'occupe chaque lettre de la chaine passée en paramètre.

```
## Renvoies le nombre de fois ou apparaissent les lettres dans l'ensemble names
nbLettres <- function(names)
{
        nbLettre = matrix(rep(0,26*length(names)),nrow=1, ncol=26);
        for (i in 1:length(names))
        {
            c = alphabetIndex(names[i]);
            nbLettre[1,c] <- nbLettre[1,c]+1;
        }
        nbLettre;
}
```

La matrice mat contient une matrice indiquant pour chaque mot si une lettre est présente ou pas. Chaque ligne de la matrice correspond à une lettre de l'alphabet, chaque colonne un mot. Par exemple, si la lettre 'a' est presente dans le mot numero 3, alors dans la case (1,3) de la matrice nous aurons la valeur 1, 0 sinon.

Question 4

Nous obtenons le tableau ci-dessous qui représente le nombre d'occurences des lettres (1-26) présentes dans l'ensemble des noms.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
150	36	86	33	184	15	41	49	124	2

11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
5	82	53	118	140	61	12	130	53	82

21	**22**	**23**	**24**	**25**	**26**
107	19	1	4	13	5

Nous observons donc que la lettre i (9) apparait 124 fois. La lettre apparaîssant le plus de fois est la lettre e (5), avec 184 occurences.

Question 5

```
#Calcul de l'entropie
#log2(t^t)
calculEntropie <- function(indexLettre, names)
{
  n = length(names);
  nl = nbLettres(names)[1,indexLettre]

  entropie <- log2((nl/n)^(-nl/n)) - log2(((n-nl)/n)^((n-nl)/n));
  entropie;
}
```

L'entropie max est obtenue pour la lettre o (15) avec une entropie de 0.9999189.

Question 6

La première question doit être le max des entropies pour avoir la lettre qui revient le plus souvent dans l'ensemble actuel.

```
#premiere question
firstQuestion <- function(names)
{
  mat = containsLetter(names);
  entropies = matrix(rep(0,26),nrow=1,ncol=26);
  props = matrix(rep(0,26),nrow=1,ncol=26);
  for (i in 0:26)
  {
    entropies[1,i] <- calculEntropie(i,names);
  }
  entropies;
}
#On recuperera le max dans d'autres fonctions avec, questionTab <- firstQuestion(
    names);indiceLettre <- which.max(questionTab);
```

En utilisant cette fonction on observe que la lettre ayant la plus grande entropie sur notre liste de noms est la lettre *o*. Le nombre d'occurences d'une lettre parmis un ensemble de mots n'est donc pas l'information à considérer dans ce genre d'exercice. Nous aurons donc intérêt à utiliser l'entropie et non le nombre d'occurences comme facteur discriminant pour minimiser le nombre de proposition pour trouver le mot rapidement.

Question 7

```
#partage
partage <- function(names)
{
  questionTab <- firstQuestion(names);
  indiceLettre <- which.max(questionTab);
  partition <- sum(containsLetter(names)[indiceLettre,]);
  partition2 <- length(names) - sum(containsLetter(names)[indiceLettre,]);
  cpt <- 1;
  cpt2 <- 1;
  for (i in 1 : length(names))
  {
    if (identical(TRUE, (containsLetter(names)[indiceLettre, i]) == 1)) {
      partition[cpt] <- names[i];
      cpt <- cpt +1 ;
    } else {
      partition2[cpt2] <- names[i];
      cpt2 <- cpt2 +1 ;
    }
  }           return(list('lettre'=indiceLettre,'partition1'=partition,'partition2'=partition2));
}
```

Dans cette fonction nous récuperons l'indice de la lettre qui a la plus grande entropie sur l'ensemble names. Nous créons deux partitions, nous parcourons l'ensemble et si la lettre avec l'entropie max est contenue dans un mot, nous rangeons ce mot dans la partition1 sinon dans la partition2. Cela nous permet de diviser l'ensemble. Autrement dit, c'est dans cette fonction que nous pouvons binariser un noeud de notre arbre. Nous retournons trois éléments, l'indice de la lettre avec l'entropie max, la partition contenant les mots avec la lettre entropie max et la deuxième partition contenant les autres mots.

Question 8

```
#partageIteratif
partageIteratif <- function(names,cpt)
{
  ensemble <- partage(names);
  ens <- names;
  if(length(names)>1 && (calculEntropie(ensemble$lettre, names)>0.01)) {
    part1 <- partageIteratif(ensemble$partition1,cpt+1);
    part2 <- partageIteratif(ensemble$partition2,cpt+1);
    ens <- list(part1, part2);
  }
  return(ens);
}
```

Cette fonction va créer notre arbre de décision. On donne en entrée un paramètre, un ensemble de base, et la fonction va récursivement découpé cet ensemble de base en sous ensemble jusqu'à ce que toute les feuilles de l'arbres, autremement dit un sous ensemble de un mot, soit atteint. Nous nous servirons du retour de cette fonction pour dessiner l'arbre.

Question 9

```
#interactif
interactif <- function(names)
{
  print("Saisir un mot a devinner dans l'ensemble donne:");
  mot <- scan("",what="character",nlines=1);
  print("Mode auto: (y/n)");
  mode <- scan("",what="character",nlines=1);
  if(mode == "y"){
    jeuAuto(names, mot);
  } else {
    jeuManuel(names);
  }
}

jeuManuel <- function(names)
{
  if(length(names)==1){
    print("trouve: ");
    print(names)
  } else {
    ensemble <- partage(names);
    print("Ce mot contient-il la lettre: (y/n)");
    print(ensemble$lettre);
    confirm <- scan("",what="character",nlines=1);
    if(confirm == "y"){
      jeuManuel(ensemble$partition1);
    } else {
      jeuManuel(ensemble$partition2);
    }
  }
}

jeuAuto <- function(names, mot)
{
  if(length(names)==1){
    print("trouve: ");
    print(names)
  } else {
    ensemble <- partage(names);
    containLettre <- containsLetter(mot);
    if(containLettre[ensemble$lettre,]==1){
      jeuAuto(ensemble$partition1, mot);
    } else {
      jeuAuto(ensemble$partition2, mot);
    }
  }
}
```

Dans cette partie de code R nous avons réaliser un déroulement de jeux, l'utilisateur saisie un mot qui devra être deviné par l'ordinateur. Le mode est une simple option, si on choisis le mode manuel l'oridnateur nous demandera de saisir si oui ou non un caractère est présent dans notre mot, autrement l'ordinateur obtiendra cette réponse en fesant appel à la méthode *containsLetter(ensemble)* par lui même.

Conclusion

Nous avons observé et mis en pratique le calcul de l'entropie. L'entropie est un indice qui nous indique l'importance, le poids de cette donnée. Exemple, nous avons constaté que la lettre *e* est la plus utilisée dans notre liste de mots mais que la lettre *o* nous permet d'obtenir une information plus grande (discriminer au mieux) par rapport aux autres lettres, son entropie se rapproche de 1 et est la valeur la plus grande parmis les entropies des autres lettres. Nous avons ensuite utilisé l'entropie pour obtenir un nombre de proposition minimum pour trouver le mot voulus de manière rapide.