TP - RdF: Arbres de décision

François Lepan - Benjamin Van Ryseghem

26 mars 2013

Introduction

Ce rapport fait état de l'utilisation et de la compréhension d'arbre de décisions. Pour y arriver nous passerons d'abord par des questions de bon sens suivi du jeu du pendu.

1 Question de bon sens

1.1 Que vaut N ?

N vaut 16 car pour quatre propositions on à toujours que deux possibilités (dichotomie).

Explication : Si en quatre propositions il peut trouver la solution c'est qu'il y a 4 niveaux dans l'arbre et pour chaque niveau il y a deux possibilités. Il a donc $2^4 = 16$ possibilités (*cf.* Fig. 1).

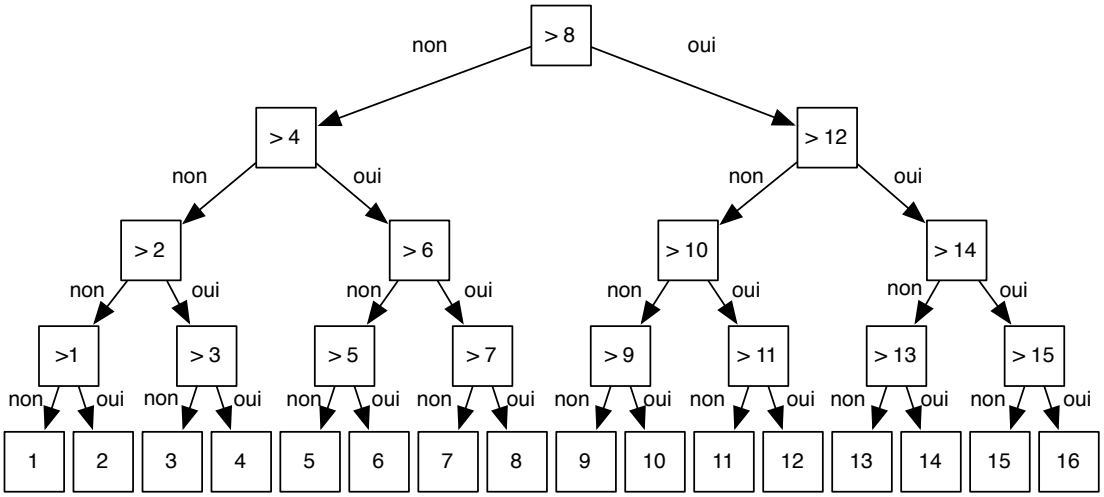

Figure 1 –

1.2 Expliquer le raisonnement de B et calculer la quantité d'information que lui a donnée A par sa réponse

En connaissant N et sachant qu'il faudra 4 propositions pour gagner, on peut facilement en déduire que la réponse se trouve dans les feuilles (1, 3, 5, 7, 9, 11, 13, 15) et donc il arrivera à trouver la solution en 3 propositions.

2 Variante du jeu du pendu

2.1 But du jeu

Si le programme est sûr de trouver la solution en au plus p questions, quelle est la relation entre n et p ? En déduire une valeur numérique minimale pour p.

On sait que pour ce jeu on aura un arbre binaire de choix. On aura donc comme relation $N = 2^p + Reste$. On sait que N = 283 qui n'est pas un multiple de 2. On en déduit que p va se trouver entre $2^8 = 256$ et $2^9 = 512$. Mais pour être sur de trouver une solution il faut choisir p = 9 sinon certaines questions n'auront pas de réponse.

2.2 Que fait ce code ?

```
mat = zeros(35,n);

for i = 1 : n
        c = str2code(noms(i));
        mat(c,i) = 1;
end
```

Ce code stocke dans *mat*, en colonnes, les lettres du mot qui sont présentes (35 car "a" == 10). Pour cela on met 1 si la lettre est présente (si elle est là deux fois on ne le saura pas).

2.3 Quelle lettre est la plus représentée ?

```
// retourne une matrice de 283 colonnes et 35 lignes contenant
// pour chaque colonne un 1 ou un 0 si il contient la lettre correspondant à
// l'indice (a à pour indice 10)
function mat = presentLettersIn(noms)

    n = size(noms,1);

    mat = zeros(35,n);

    for i = 1 : n
        c = str2code(noms(i));
        mat(c,i) = 1;
    end

endfunction
```

```
function h = getNumberOfLetters(noms)

    // on crée une colonne de 283 afin de faire une multiplication
    // entre cette colonne et la matrice de 1 et 0 retournée par presentLettersIn
    // afin de récupérer une seul colonne de 35 contenant pour chaque élément
    // la somme des éléments de sa ligne.
    one = ones(size(noms,1),1);

    m = presentLettersIn(noms)

    h = m * one;

endfunction


function letter = mostPresentLettersIn(noms)

    h = getNumberOfLetters(noms);

    // On récupère l'indice de la lettre la plus présente
    [value, mostPresentLetterCode] = max(h);

    // on transforme l'indice en lettre
    letter = code2str(mostPresentLetterCode);

endfunction

// Chargement de la base de noms d'animaux
exec("rdfAnimaux.txt");

// on récupère la lettre la plus présente dans tous les mots
letter = mostPresentLettersIn(noms)
```

Après exécution du code on obtient la lettre e.

2.4 Entropie

```
function resultat = entropie(noms, n)

    m = getNumberOfLetters(noms);

    resultat = - log((m./n).^(m./n)) - log((1 - m./n).^(1 - m./n))

endfunction
```

Voici la fonction qui permet de calculer l'entropie

2.5 Que fait ce code ?

```
I=([1:n]>0);

[A,B,i]=partage(I);code2str(i),sum(A),sum(B)
[C,D,j]=partage(A);code2str(j),sum(C),sum(D)
[E,F,k]=partage(B);code2str(k),sum(E),sum(F)
```

Il affiche la répartition des mots dans l'arbre. Pour le premier partage on a 140 mots dans A et 143 dans B. Pour le partage de A on obtient deux ensembles C contenant 71 mots et D 69 mots. Et enfin pour le partage de B on obtient deux ensembles C contenant 70 mots et D 73 mots.

Si on continue on aura pour chaque noeud le nombre de fils qu'il possède, pour chacun de ses fils le nombre de fils, etc.

2.6 L'arbre

Profondeur

Afin de connaître la profondeur maximale de l'arbre il suffit de rajouter ces lignes en bleu à la fonction *arbre* :

```
function maxim = arbre(I,str)
    if (sum(I)>1) then

        [A,B,i] = partage(I);
        max1 = arbre(A,str+'| ');
        printf("%s%c (%i,%i)",str,code2str(i),sum(A),sum(B));
        max2 = arbre(B,str+'| ');

        maxim = 1 + max(max1,max2);

    else
        printf("%s -> %s",str,noms(I));
        maxim = 0;
    end

endfunction
```

Le résultat est 10.

Mots défavorables

Afin d'afficher les mots les plus défavorables il faut rajouter ces lignes en bleu à la fonction *arbre* :

```
function maxim = arbre(I,str,currentDeepth,maxDeepth)

    if (sum(I)>1) then

        currentDeepth = currentDeepth +1;

        [A,B,i] = partage(I);

        max1 = arbre(A,str+'| ',currentDeepth,maxDeepth);

        printf("%s%c (%i,%i)",str,code2str(i),sum(A),sum(B));

        max2 = arbre(B,str+'| ',currentDeepth,maxDeepth);

        maxim = 1 + max(max1,max2);

    else

        if currentDeepth == maxDeepth then
            global deepestElements;
            deepestElements = [deepestElements noms(I)];
        end

        printf("%s -> %s",str,noms(I));
        maxim = 0;
    end

endfunction
```

Ainsi qu'une variable globale : *global deepestElements ;*

```
global deepestElements;
// pour le vider à chaque exécution
deepestElements = [];

maximum = arbre(I,' ',0,10)
```

L'exécution du code précédent nous fournit les animaux suivants : roussette, tortue, hareng, panthere, merlan et varan.

Moyenne

Afin d'avoir en plus le nombre moyen nécessaire pour trouver la solution en suivant cet arbre il faut ajouter les lignes en bleu à la fonction *arbre* :

```
function [nbQuestion, maxim] = arbre(I,str,currentDeepth,maxDeepth)

    if (sum(I)>1) then

        currentDeepth = currentDeepth +1;

        [A,B,i] = partage(I);

        [nbQuestion1, max1] = arbre(A,str+'| ',currentDeepth,maxDeepth);

        printf("%s%c (%i,%i)",str,code2str(i),sum(A),sum(B));

       [nbQuestion2, max2] = arbre(B,str+'| ',currentDeepth,maxDeepth);

        maxim = 1 + max(max1,max2);
        nbQuestion = nbQuestion1 + nbQuestion2;

    else

        if currentDeepth == maxDeepth then
            global deepestElements;
            deepestElements = [deepestElements noms(I)];
        end

        nbQuestion = currentDeepth;

        printf("%s -> %s",str,noms(I));
        maxim = 0;
    end

endfunction
```

ainsi que diviser le nombre total de question par le nombre d'animaux.

```
[nbQuestion, maximum] = arbre(I,' ',0,10);

moyenne = nbQuestion / n
```

On obtient comme réponse 8.2756184. Cette différence avec p s'explique car $2^8 = 256 < n = 283 < 2^9 = 512$ donc il est normal de trouver une réponse entre 8 et 9.

Conclusion

Au vue de ces résultats nous pouvons dire que les arbres de décisions sont un moyen efficace de trouver des solutions en un temps record et leur mise en oeuvre est facile.