

COMPTE-RENDU TP9

Reconnaissance de formes

BADACHE Yassine – DHERSIN Jérôme

Introduction :

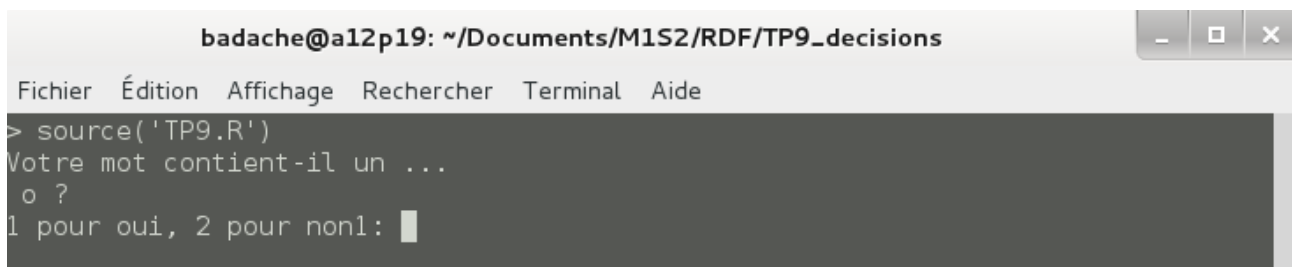
L'objectif de ce TP est de réaliser une variante du jeu du pendu, en utilisant les méthodes relatives aux arbres de décision afin de trouver un mot parmi une liste prédéfinie. Ce compte-rendu liste les différents résultats, moyens d'utilisation et contraintes liées à l'utilisation de notre script R.

Langage.

Le langage imposé est le langage R, qui est un langage particulièrement adapté pour le calcul et les opérations matricielles.

Utilisation.

Pour activer le jeu, lancez l'interface R (chemin brut *opt/R-3.0.1/bin/R* sur les machines de la faculté) et entrez `source("TP9.R")`. Le programme attend deux types de réponses qui vous seront indiquées, comme le montre la capture d'écran ci-dessous :



```
badache@a12p19: ~/Documents/M1S2/RDF/TP9_decisions
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
> source('TP9.R')
Votre mot contient-il un ...
o ?
1 pour oui, 2 pour non1: █
```

Le système attend un entier, soit 1 dans le cas où le mot auquel vous avez pensé contient la lettre, soit 2 dans le cas où elle ne la contient pas. Toute autre tentative se soldera par un échec et une interruption du script, alors faites bien attention !

Le jeu s'arrêtera automatiquement au moment où il sera sûr d'avoir trouvé le mot, en utilisant la méthode décrite ci-après. Il vous affichera alors le mot en question, et vous serez surpris de voir qu'il a correctement trouvé, seulement via la liste des lettres que le mot contient ou non.

N.B : La liste fournie avec le jeu de base est une liste de 283 animaux. Il est donc obligatoire de choisir un animal, un nom non composé, et existant afin de jouer correctement. De même, vous pouvez changer la liste à tout moment et la remplacer par une liste de mots de n'importe quelle taille, en respectant la syntaxe du fichier fourni avec l'application.

Fonctionnement.

Afin de fonctionner au mieux, le script utilise une méthode qui peut s'apparenter à une recherche dichotomique. Elle utilise les principes **d'entropie** d'une variable (*pour résumé, la plus grande entropie représente la lettre séparant au mieux une liste en deux*) pour calculer la variable optimale qui permettrait de séparer une liste en deux parties à peu près égales.

Comme on peut le constater sur le screenshot, la première lettre est la lettre 'o'. C'est donc elle qui permet de séparer le mieux la liste en deux parties distinctes. Pour ne pas ôter le plaisir de jeu, voici la séparation par la lettre 'o', les autres séparations ne vous seront alors pas révélées.

[1]	"albatros"	"alligator"	"alouette"	"anaconda"
[5]	"anchois"	"antilope"	"aoutat"	"asticot"
[9]	"babouin"	"bison"	"boa"	"boeuf"
[13]	"bouc"	"bouquetin"	"bulot"	"cacatoes"
[17]	"cachalot"	"cameleon"	"caneton"	"caribou"
[21]	"castor"	"chamois"	"chouette"	"cigogne"
[25]	"cloporte"	"cobra"	"coccinelle"	"cochon"
[29]	"colibri"	"coq"	"corbeau"	"corneille"
[33]	"coucou"	"couleuvre"	"coyote"	"crocodile"
[37]	"dindon"	"doryphore"	"dragon"	"dromadaire"
[41]	"escargot"	"espadon"	"esturgeon"	"etourneau"
[45]	"faon"	"faucon"	"fouine"	"fourmi"
[49]	"frelon"	"gnou"	"goeland"	"gorille"
[53]	"grenouille"	"grillon"	"guenon"	"herisson"
[57]	"heron"	"hibou"	"hippocampe"	"hippopotame"
[61]	"hirondelle"	"homard"	"kangourou"	"koala"
[65]	"labrador"	"lamproie"	"langouste"	"leopard"
[69]	"lion"	"lionceau"	"lionne"	"loir"
[73]	"lombric"	"lotte"	"loup"	"loutre"
[77]	"louve"	"luciole"	"lycaon"	"manchot"
[81]	"mangouste"	"marabout"	"marmotte"	"marsouin"
[85]	"moineau"	"morse"	"morue"	"mouche"
[89]	"mouette"	"mouflon"	"moustique"	"mouton"
[93]	"oie"	"oiseau"	"okapi"	"ornithorynque"
[97]	"orque"	"oryx"	"otarie"	"ouistiti"
[101]	"ours"	"oursin"	"paon"	"papillon"
[105]	"pekinois"	"percheron"	"perroquet"	"petoncle"
[109]	"phacochere"	"phoque"	"pigeon"	"pingouin"
[113]	"poisson"	"poney"	"porc"	"pou"
[117]	"poulain"	"poule"	"poulet"	"poussin"
[121]	"pucceron"	"putois"	"python"	"ragondin"
[125]	"raton"	"rhinoceros"	"rossignol"	"roussette"
[129]	"rouget"	"siamois"	"sole"	"souris"
[133]	"tamanoir"	"taon"	"thon"	"tortue"
[137]	"toucan"	"triton"	"vautour"	"vison"

Le système, une fois qu'il a séparé la liste, prendra cette liste pour ré-itérer ses opérations en recalculant l'entropie de chaque lettre et proposer à l'utilisateur la lettre la plus pertinente.

Feature manquante.

Toute la partie concernant les arbres n'a pas été traitée. En effet, l'arbre n'est pas construit ni affiché en console, bien que le jeu, lui, soit fonctionnelle et applique correctement les principes d'entropie et de recherche de paramètre optimal de séparation .

Conclusion.

Ces travaux pratiques nous ont permis d'appliquer les principes de construction d'arbre et d'entropie de manière ludique, construisant ainsi un jeu fonctionnel aux règles claires et infaillible, et ce uniquement par le calcul. C'est une approche que nous n'avons pas l'habitude d'adopter pour quelque chose d'interactif et de divertissant et cela a été selon nous une des manières les plus intéressantes pour apprendre des notions qui peuvent paraître au premier abord rébarbatives, comme l'application de formules ou de notions.

ANNEXE : Code de la macro R.

```
# Chargement de la base de noms d'animaux
source ("rdfAnimaux.txt")

# Convertit une chaîne de caractère en nombre (ASCII)
str2int <- function(x)
{
    strtoi(charToRaw(x),16L)-96
}

# Convertit un int en char
int2str <- function (x)
{
    x <- x + 96
    mode (x) <- "raw"
    return (rawToChar(x))
}

lettre_informative <- function (liste_noms)
{
    nb_max <- 0
    mat = matrix(rep(0, 26*length(liste_noms)),nrow=26, ncol=length(liste_noms))
    colnames(mat) = liste_noms

    for (name in liste_noms)
    {
        c = str2int(name);
        mat[c,name] <- 1;
    }

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

    # Application de la fonction du logarithme
    h <- -(log(h^h)) - (log ((1-h)^(1-h)))

    lettre <- which.max(h)

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

```

jouer <- function (list)
{
  trouve = FALSE
  solution <- ""

  while (!trouve)
  {
    list <- lettre_informative(list)

    cat("Votre mot contient-il un ...\n", intToUtf8(list$let + 96), "? \n1 pour oui, 2 pour
non\n");

    choix = scan(what="int", nmax = 1);

    if (choix_joueur == 1)
    {
      list = list$avec
      print (list)
    }

    else if (choix_joueur == 2)
    {
      list = list$sans
      print (list)
    }

    if (length(list) == 1)
    {
      solution = list[1]
      print (solution)
      trouve = TRUE
    }

  }

  cat ("Vous cherchiez ...", solution, " ?\n");
}

```

```

# A vous de jouer !
jouer (noms)

```