Challenge R 2021

« Aide à la décision »

Andrew Mary Huet de Barochez
Xavier Lemaire

I. Introduction

A. Objectif

Le but de ce challenge est de créer un algorithme capable de classifier des pages web (fichier html) dans 7 classes différentes.
Ces classes sont : Accueil, blog, commerce, FAQ, home, liste et recherche.
Les étudiants possèdent 150 fichiers par classe pour un total de 1050 fichiers et doivent entraîner leur classificateur sur ces derniers.

Les classificateurs sont testés par les enseignants sur 350 fichiers dont les élèves n'ont pas accès et la note est évaluée en fonction du taux de classification des pages.

B. Modalités

Les étudiants peuvent soumettre 5 dépôts contenants :

- un fichier main.r contenant les différentes fonctions de traitement de données ainsi que d'une fonction classer prenant en paramètre un chemin de fichier et retournant la prédiction de classe de ce fichier.
- un fichier readme expliquant la démarche utilisé
- les ressources nécessaires au fonctionnement du classificateur

Ces dépôts seront notés en fonction de la performance du classificateur et la meilleure note sera gardée.

C. Exemple de donnée

Figure 1 : exemple de donnée, page html classique.

II. Traitement des données

A. Suppression de balises HTML / JS

Le traitement des données est une étape cruciale pour faire fonctionner convenablement un classificateur. Il faut que les données passées en paramètre puissent apporter de l'information dans le but de discriminer les différentes classes et ainsi obtenir de bonnes prédictions.

Nous avons déduit par l'intuition puis par le calcul, que les balises de code ne permettraient pas de discriminer les documents. En effet elles seront forcément communes à toutes les pages (même si l'on pourrait imaginer que dans la classe "liste" il y est plus de balises <ul>), en pratique nous obtenons 7% d'erreurs en plus en gardant les balises HTML.

B. Fouille de texte

Après avoir supprimé toutes les balises de code, nous obtenons du texte brut. On effectue un traitement préalable qui va nous faciliter la tâche pour les prochaines étapes. Ce traitement consiste à convertir toutes les majuscules en minuscules, car deux mots avec une majuscule ou non portent la même information sauf pour certaines exceptions (Homme avec un grand H =/= homme).

Il faut tout d'abord comprendre que les mots ne possèdent pas tous le même taux d'information. Par exemple, si nous prenons le pronom "she" : il peut apporter une information sur le genre du nom d'après mais n'apporte pas concrètement d'information sur les classes que nous souhaitons obtenir.

Prenons maintenant le mot "Teaching", il s'agit d'un verbe indiquant une action précise qui peut être liée à une de nos classes. On applique donc une suppression des mots d'arrêts qui n'apportent pas réellement d'information.

Pour encore améliorer le traitement et aider à la classification, nous appliquons une méthode de racinisation permettant de ne garder que la racine des mots, donc son idée initiale. Avec "Teaching" on obtient "Teach", et ainsi "Teacher", "Teaching" et "Teach" donnent tous "Teach" permettant de regrouper ces mots en une idée commune qui sera un paramètre de notre classificateur.

Pour finir nous supprimons les chiffres car nous n'avons pas trouvé qu'il s'agissait d'un discriminant très intéressant dans nos différentes classes.

C. Filtrage des mots par leur fréquence

Cette partie vient en complément à la suppression des mots d'arrêts, on va choisir des valeurs minimum et maximum de fréquence d'un mot pour qu'il soit gardé comme paramètre de notre classificateur ou non.

Une fréquence maximum pour que les mots trop utilisés, donc similaires à des mots d'arrêts, soient supprimés, et une fréquence minimum pour que les mots trop spécifiques qui font référence à un événement particulier et non pas à une caractéristique redondante de classe soient également supprimés. Pour choisir ces valeurs de fréquence nous avons testé différentes valeurs en regardant l'impact de ce changement sur le pourcentage de bonne classification, voir figure 1 et 2 qui ont utilisé un classificateur de forêts d'arbres décisionnels que nous verrons plus tard.

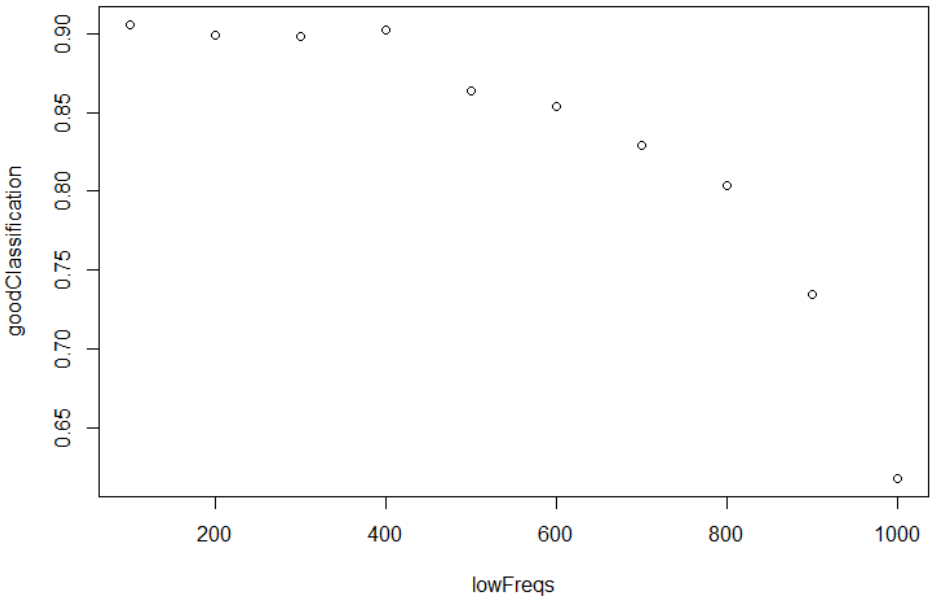

Figure 2 : évolution du taux de bonne classification en fonction de la fréquence minimum et maximum d'un mot pour qu'il soit inclu en paramètre du classificateur. La fréquence maximum est fixée à 1400 occurrences et la minimum de 100 à 1000 occurrences avec un pas de 100.

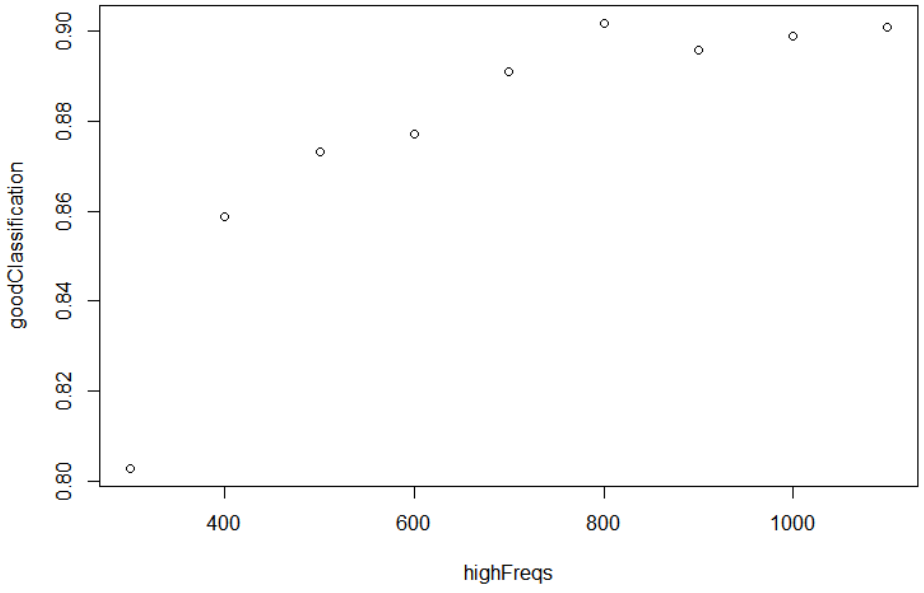

Figure 3 : évolution du taux de bonne classification en fonction de la fréquence maximum d'un mot pour qu'il soit inclu en paramètre du classificateur. La fréquence minimum est fixée à 200 occurrences et la fréquence maximum augmente de 400 à 1200 occurrences avec un pas de 100.

III. Classifieur K plus proche voisin (KPPV)

A. Principe

Ce classificateur à été réalisé de zéro avec l'aide de notre enseignant. Il calcule la distance entre chaque paramètre d'entrée et grâce aux observations récoltées lors de la phase d'entraînement, calcule un ensemble de taille K classes qui sont le plus proche des paramètres d'entrée et retourne la classe qui est en majorité dans cet ensemble.

B. Implémentation

```
dist_voisins←function(vecteur,data){
  return(apply(data[,-ncol(data)],1,distanceEuclidienne,x=vecteur))
}

kppv←function(vecteur, k, data){
  v←dist_voisins(vecteur,data)
  return(order(v)[1:k])
}

classerKPPV←function(vecteur, k, data) {
  vec←kppv(vecteur, k, data)
  classes←data[vec,ncol(data)]
  occ←table(classes)
  return (names(occ)[which.max(occ)])
}
```

dist_voisins : retourne la distance entre chaque élément d'un vecteur et chaque ligne de la matrice où l'on stocke nos mots et leurs fréquence pour chaque page de chaque classe.
kppv : retourne un nombre de K classes qui sont proche du vecteur passé en paramètre.
ClasserKPPV : retourne le nom de la classe la plus présente dans le vecteur de retour de la fonction kppv.

C. Les différents calculs de distance

Pour calculer la distance dans la méthode KPPV nous avons utilisé différentes méthodes et il s'avère que dans notre cas la distance Euclidienne nous donne de meilleur résultat avec k=4 comme le montre le tableau ci-dessous.

Nom du calcul de distance	Taux d'erreur
Distance Euclidienne	0.2247619
Distance de Manhattan	0.3180952
Distance de MinKowski avec p = 0.25	0.84
Distance de Jaccard	0.9447619

Tableau 1 : taux d'erreur par méthode de distance utilisée en sachant que k = 4.

D. Validation des résultats

Afin de valider les résultats de manière fiable nous avons créé une fonction permettant de vérifier la classe de chaque document du corpus tout en utilisant un classificateur n'ayant pas été entraîné avec le document qui est en train d'être vérifié.

```
erreurKPPV←function(k, data) {
  res←sapply(1:nrow(data),
             function(i) classerKPPV(data[i,-ncol(data)],
                                     k,
                                     data[-i,]))
  return (1-sum((res==(data[,ncol(data)]))/nrow(data)))
}
```

Nous obtenons 79,01% de bonne classification sur nos données.

E. Résultats avec les données de l'enseignant

accueil	blog	commerce	FAQ	home	liste	recherche	total
1.00	0.90	0.78	1.00	0.86	0.26	0.76	0.79

Tableau 2 : taux de bonne classification avec la méthode KPPV.

Avec la distance Euclidienne, une lowFreq de 400, une highFreq de 1400 et un k égal à 4 nous obtenons 79% de bonne classification. Nous avons essayé d'autres paramètres n'ayant pas abouti donc nous avons décidé que la méthode k plus proche voisin n'est pas assez adaptée à ce problème. Nous allons donc nous pencher vers les réseaux de neurones artificiels pour le prochain classificateur.

IV. Réseau de neurones artificiels

A. Principe

Les réseaux de neurones artificiels sont des systèmes inspirés des neurones biologiques et se basent sur des méthodes statistiques. Ce modèle comporte trois couches différentes : la couche d'entrée qui comporte autant de neurones que de paramètres, la couche de sortie qui compte un neurone par classe à prédire et les couches cachées qui servent à stocker des biais et à orienter chaque paramètre vers une classe.

Nous avons utilisé une approche supervisée des réseaux de neurones permettant d'ajuster les différents biais dans les neurones en fonction des résultats que trouve le modèle grâce à la labellisation.

B. Implémentation

Nous avons utilisé la librairie neuralnet disponible sur le site du CRAN.

```r
# On bind à nos données les labels de classes sur 7 colones diférentes
trainData←cbind(trainData[,-ncol(trainData)],
                nnet::class.ind(as.factor(trainData$classes)))

names(trainData)←c(names(trainData)[1:(ncol(trainData)-7)],
                   "l1", "l2", "l3", "l4", "l5", "l6", "l7")

# Création de la formule
n←names(trainData)
f←as.formula(paste("l1 + l2 + l3 + l4 + l5 + l6 + l7 ~",
             paste(n[!n %in% c("l1","l2","l3","l4","l5","l6","l7")],
             collapse = " + ")))

# Instanciation et paramètrage du modèle
nn=neuralnet(f,
             data = trainData,
             hidden = 30,
             act.fct = "logistic",
             linear.output = FALSE,
             lifesign = "minimal",
             stepmax = 300)


# Apprentissage du modèle
pr.nn ← compute(nn, trainData[,1:(ncol(trainData)-7)])
```

Pour préciser au modèle les classes à prédire et les paramètre il faut écrire la formule qui prend cette forme :

classe 1 + … + classe n ~ param1 + … + param n

Pour les paramètres du modèle nous avons mis qu'une couche cachée contenant 30 neurones car nous avons de meilleurs résultats, la fonction d'activation est une fonction logistique et nous avons limité le modèle à 300 epochs.

C. Validation des résultats

Afin de valider nos résultats nous avons utilisé la méthode de validation croisée, qui consiste à entraîner le modèle qu'avec une partie des données et de le tester avec l'autre et ce plusieurs fois en découpant différentes parties. Nous obtenons 89% de bonne classification avec cette méthode de validation.

```r
set.seed(500)
# 10 fold cross validation
k <- 10
# Results from cv
outs <- NULL
# Train test split proportions
proportion <- 0.5

for(i in 1:k)
{
  index <- sample(1:nrow(trainData),
                  round(proportion*nrow(trainData)))
  train_cv <- trainData[index, ]
  test_cv <- trainData[-index, ]
  nn_cv <- neuralnet(f,
                     data = train_cv,
                     hidden = 30,
                     act.fct = "logistic",
                     linear.output = FALSE,
                     stepmax = 300,
                     lifesign = "minimal")

  # Compute predictions
  pr.nn <- compute(nn_cv, test_cv[,1:(ncol(trainData)-7)])
  # Extract results
  pr.nn_ <- pr.nn$net.result
  # Accuracy (test set)
  original_values <- max.col(test_cv[,(ncol(trainData)-6):ncol(trainData)])
  pr.nn_2 <- max.col(pr.nn_)
  outs[i] <- mean(pr.nn_2 == original_values)
}
mean(outs)
```

D. Résultats avec les données de l'enseignant

accueil	blog	commerce	FAQ	home	liste	recherche	total
0.78	0.84	0.72	0.90	0.70	0.66	0.70	0.75

Tableau 3 : taux de bonne classification avec la méthode des réseaux de neurones.

Les résultats finaux sont très bas par rapport à notre validation croisée et notre enseignant nous a expliqué qu'il s'agit ici d'overfitting. En effet le modèle s'est beaucoup trop entraîné sur nos données et à donc du mal à se généraliser avec de nouvelles données. Ce problème arrive généralement quand il y a trop de couches et de neurones, mais dans notre cas cela n'a pas l'air d'être notre problème ayant qu'une couche cachée et 30 neurones.

Cela vient également du fait que nous avions choisi une répartition des données croisée d'une hauteur de 95% or, les données à notre disposition étant trop peu nombreuses, la validation avait tendance à être biaisée. En réduisant ce seuil à 50% nous obtenons 79% de bonne classification qui est bien plus proche du résultat obtenu avec les données de l'enseignant.

Nous nous sommes donc tournés vers une méthode impliquant moins d'overfitting.

V. Forêts d'arbres décisionnels (Random Forest)

A. Principe

L'algorithme des forêts d'arbres décisionnels génère des arbres décisionnels de manière aléatoire et garde les arbres donnant de bonnes classifications. Pour construire l'arbre final il faut donc que les données soit labellisée afin de vérifier si les différentes décisions mènent bien aux bonnes classes.

B. Implémentation

La librairie utilisée est randomForest disponible sur le site du CRAN.

```r
# Conversion des données de la colonne classes comme facteur
data$classes ← as.factor(as.numeric(data$classes))

# Définition du modèle
model ← randomForest(classes ~ ., data = data, ntree = 200)
model

predValid ← predict(model, data, type = "class")

# Vérification des prédictions
mean(predValid == data$classes)
table(predValid, data$classes)
```

La définition d'une random forest est bien plus simple que celle d'un réseau de neurones, ici nous passons notre formule avec la colonne 'classes' comme résultat et '.' pour prendre toutes les autres colonnes comme paramètres. Nous précisons également le nombre d'arbres maximum, mais ce n'est pas obligatoire.

```
> mean(predValid == data$classes)
[1] 0.9980952
> table(predValid, data$classes)

predValid   1   2   3   4   5   6   7
        1 150   0   0   0   0   0   0
        2   0 150   0   0   0   0   0
        3   0   0 149   0   0   0   0
        4   0   0   0 150   0   0   0
        5   0   0   0   0 150   0   0
        6   0   0   0   0   0 149   0
        7   0   0   1   0   0   1 150
```

C. Validation des résultats

Nous avons utilisé le même principe de validation croisée vu avec les réseaux de neurones. On obtient 0.9201905 de bonne classification mais ce taux peut varier avec l'aléatoire des forêts d'arbres décisionnels.
En faisant une simple vérification avec des sets de données différents on obtient 0.9161905.

```
> mean(predValid == ValidSet$classes)
[1] 0.9161905
> table(predValid, ValidSet$classes)

predValid  1  2  3  4  5  6  7
        1 76  0  0  0  1  2  0
        2  0 75  0  0  0  0  1
        3  0  1 64  0  1  3  2
        4  0  0  0 77  0  2  0
        5  0  0  2  0 64  6  2
        6  0  0  0  1  0 59  1
        7  0  3  4  0  6  6 66
```

D. Résultats avec les données de l'enseignant

accueil	blog	commerce	FAQ	home	liste	recherche	total
1.00	0.98	0.96	1.00	0.96	0.80	0.90	0.94

Tableau 4 : taux de bonne classification avec la méthode des forêts d'arbres décisionnels.

On obtient ici 94% de bonne classification avec des données n'ayant jamais été vues par le modèle. Les forêts d'arbres décisionnels sont dans notre cas un bon entre-deux de KPPV et des réseaux de neurones, ils combinent la précision des réseaux et la généralisation de KPPV.

Nous avons donc optimisé nos paramètres highFreq et lowFreq avec ce modèle car nous voulions trouver les meilleures valeurs pour avoir le plus de bonne classification (figure 2 et 3).

VI. Machine à vecteurs de support (SVM)

A. Principe

Tout comme les méthodes vues précédemment il s'agit d'un classificateur d'apprentissage supervisé linéaire mais pouvant aussi être non linéaire. Dans cette application nous avons utilisé un SVM radial car nous avons 7 classes à traiter et le classificateur linéaire n'avait pas l'air de répondre à nos besoins.

B. Implémentation

Nous avons utilisé la librairie e1071 disponible sur le site du CRAN.

```r
# Conversion des données de la colonne classes comme facteur
data$classes ← as.factor(as.numeric(data$classes))

# Définition du modèle
fit = svm(classes ~ ., data = TrainSet, scale = FALSE, kernel = "radial", cost = 5)

# Test de la précision du modèle
predValid = predict(fit, ValidSet)
mean(predValid == ValidSet$classes)
table(predValid, ValidSet$classes)
```

Encore une fois la définition du modèle est très simple, on spécifie qu'on ne veut pas scale nos données, qu'il s'agit du modèle radial et le paramètre de 'cost' qui permet de spécifier la "courbe de classification" : avec un 'cost' faible on vise une classification bien répartie alors qu'avec un 'cost' élevé on vise une classification plus pointilleuse d'après ce que nous avons compris.

```
> mean(predValid == data$classes)
[1] 0.9838095
> table(predValid, data$classes)

predValid   1   2   3   4   5   6   7
        1 150   0   0   0   0   0   0
        2   0 150   0   0   0   0   0
        3   0   0 146   0   0   0   0
        4   0   0   0 150   0   0   0
        5   0   0   1   0 143   1   1
        6   0   0   0   0   0 145   0
        7   0   0   3   0   7   4 149
```

C. Validation des résultats

Les validations que nous avons effectuées sont aussi les mêmes que pour les réseaux de neurones.
Nous obtenons 0.8047619 de bonne classification avec la validation croisée et 0.8 pour une validation en coupant seulement en deux le set de données.

```
> mean(predValid == ValidSet$classes)
[1] 0.8
> table(predValid, ValidSet$classes)

predValid  1  2  3  4  5  6  7
        1 75  0  0  0  0  1  0
        2  0 73  6 18  4 24  2
        3  0  0 58  0  0  1  1
        4  0  0  0 58  0  0  0
        5  0  0  0  0 50  2  1
        6  1  0  2  2  1 40  2
        7  0  6  4  0 17 10 66
```

D. Résultats avec les données de l'enseignant

accueil	blog	commerce	FAQ	home	liste	recherche	total
1.00	0.96	0.84	0.82	0.88	0.74	0.96	0.88

Tableau 5 : taux de bonne classification avec la méthode SVM radiale.

Étonnement nous arrivons à obtenir 88% de bonne classification sur les données de l'enseignant avec cette méthode alors que nos validations indiquait un taux de bonne classification 8% plus bas. La méthode SVM semble également très intéressante pour ce problème de classification mais nous n'avons pas eu le temps de pousser l'optimisation de ce modèle.

VII. Conclusion

Pour conclure, la méthode qui a eu le plus de succès est la forêt d'arbres de décisions avec 94% de bonne classification. C'est la méthode qui à su le plus s'adapter à de nouvelles données tout en ayant un très bon taux de classification.

Vient ensuite la méthode SVM avec un taux de classification de 88%, cette méthode semble adaptée à notre problème de classification mais il nous aurait fallu plus de temps afin d'en comprendre toutes les spécificités.

La méthode K plus proche voisin arrive avec 79% de bonne classification, il semblerait qu'elle soit très adaptée au fait de recevoir de nouvelles données. Cependant le modèle ne prédit pas très bien les classes quand il y a trop de paramètres 'parasites', qui ne permettent pas de discriminer convenablement, alors que d'autres méthodes comme les réseaux de neurones et les forêts d'arbres de décision en font plus abstraction.

La plus mauvaise est le réseau neuronal avec 75% de bonne classification mais ce résultat aurait pu certainement être amélioré à 80-85% en modifiant les paramètres du modèle. Nous pensons que cette méthode est très intéressante mais que dans ce cas elle n'est pas très adaptée car nous avons un nombre de données trop peu conséquent.

Type de classifieur	accueil	blog	commerce	FAQ	home	liste	recherche	total
KPPV	1.00	0.90	0.78	1.00	0.86	0.26	0.76	0.79
Réseau neuronal	0.78	0.84	0.72	0.90	0.70	0.66	0.70	0.75
Random Forest	1.00	0.98	0.96	1.00	0.96	0.80	0.90	0.94
SVM	1.00	0.96	0.84	0.82	0.88	0.74	0.96	0.88

Tableau 6 : taux de bonne classification de toutes les méthodes utilisées.

Code source du projet : https://github.com/PhoqueEberlue/r-challenge

VIII. Annexes

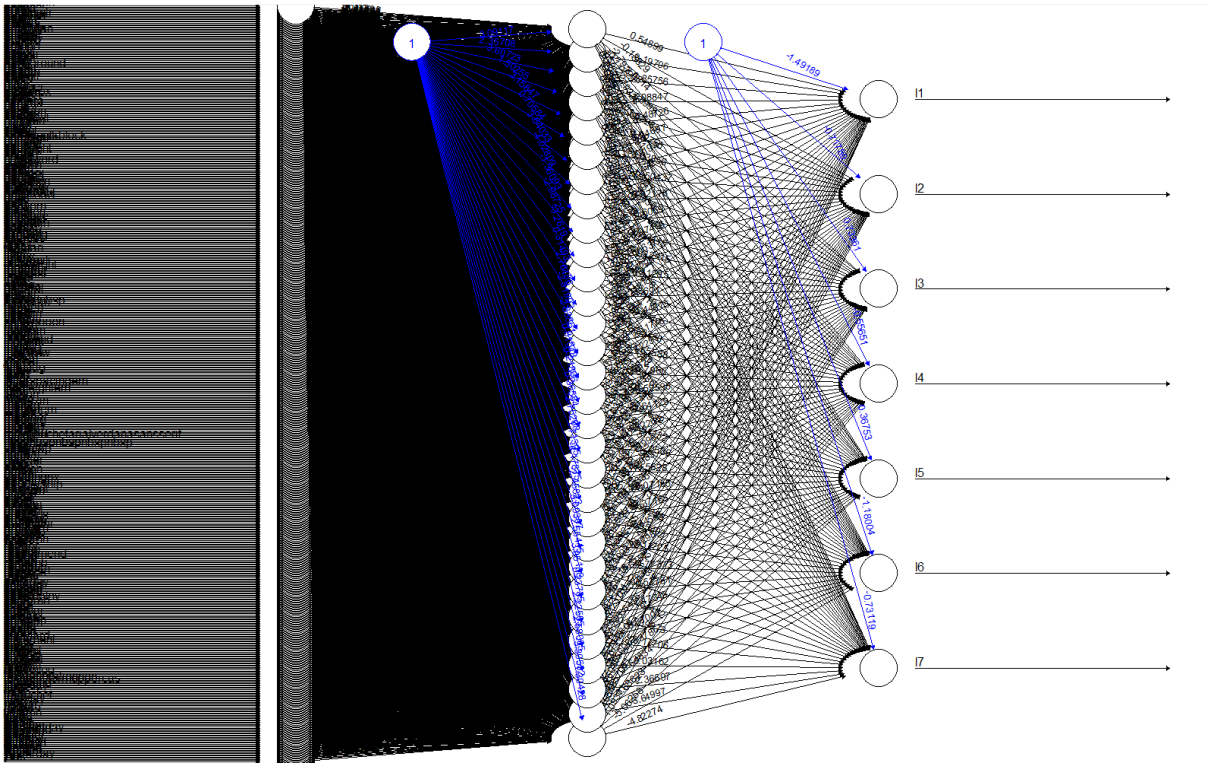

Figure 4 : Affichage de notre réseau de neurones. A gauche est affiché chaque mots qui sont en paramètre de chaque neurones d'entré, au milieu nous trouvons nos 30 neurones de la couche cachée et à droite les 7 neurones de résultat qui représente nos classes.

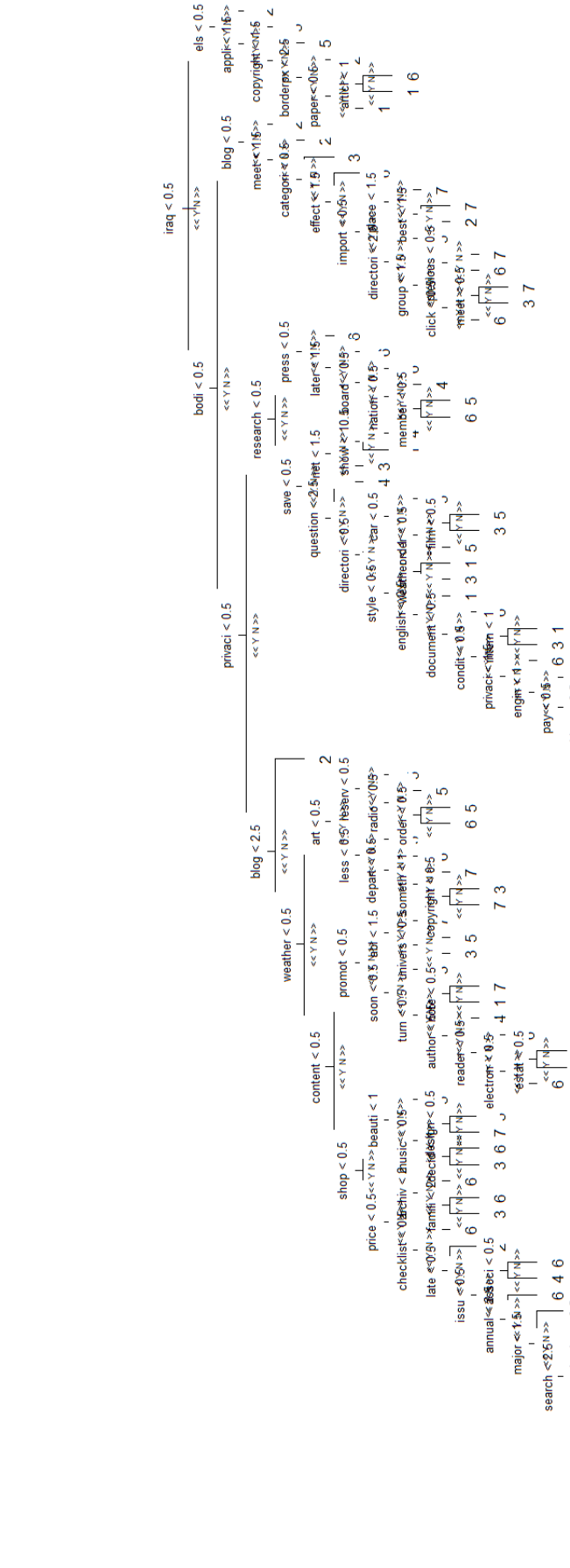

Figure 5 : affichage de l'arbe décisionnel final produit par la méthode random forest.

IX. Sources

A. Réseaux de neurones

https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf

https://www.datacamp.com/community/tutorials/neural-network-models-r

https://www.r-bloggers.com/2017/02/multilabel-classification-with-neuralnet-package/

http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/fr_Tanagra_Packages_R_for_Deep_Learning.pdf

https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw

B. Forêts d'arbre de décision

https://thinkr.fr/premiers-pas-en-machine-learning-avec-r-volume-4-random-forest/

https://www.r-bloggers.com/2018/01/how-to-implement-random-forests-in-r/

https://cran.r-project.org/web/packages/randomForest/randomForest.pdf

C. Machine à vecteurs de support

https://www.datacamp.com/community/tutorials/support-vector-machines-r

https://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf