Reconnaissance de langues écrites HMM

Mamadou KANOUTE

18/12/2020

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(mvtnorm)
library(igraph)
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:dplyr':
##
##     as_data_frame, groups, union
```

```
## The following objects are masked from 'package:purrr':
##
##     compose, simplify
```

```
## The following object is masked from 'package:tidyr':
##
##     crossing
```

```
## The following object is masked from 'package:tibble':
##
##     as_data_frame
```

```
## The following objects are masked from 'package:stats':
##
```

```
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

Exercice 1: Traitement de textes

Import du fichier csv contenant les textes

J'importe les textes à partir d'un fichier csv.
Le fichier .csv comporte **90 textes (anglais et français)**
Pour chaque ligne, le séparateur entre le texte et le y (langue du texte) est | comme suit:

texte|-1 : texte en français
texte|1 : texte en anglais

```
file = "/home/boua/Documents/Tps/2020_2021/Non_supervisé/Projet/textsbruts.csv"
readFile = read.csv(file, sep = "|")
set.seed(42)
rows <- sample(nrow(readFile))
readFile <- readFile[rows, ]
intialTexts = as.vector(readFile$Textes)
y = readFile$classes

print("Les y sont: ")
```

```
## [1] "Les y sont: "
```

```
print(y)
```

```
##  [1] -1 -1 -1  1  1  1 -1  1  1 -1  1 -1 -1 -1 -1 -1  1 -1  1 -1 -1  1 -1 -1  1
## [26]  1  1  1  1  1  1  1 -1 -1  1 -1  1 -1 -1  1  1  1  1  1  1 -1  1  1 -1 -1
## [51]  1 -1 -1 -1  1 -1  1  1  1  1 -1 -1  1 -1 -1  1  1 -1  1 -1  1  1 -1  1 -1
## [76]  1 -1  1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1 -1
```

Fonctions pour le nettoyage de texte

Dans cette partie, j'effectue les actions suivantes sur les textes:

- supprimer caractères spéciaux: [(),-,.:;-?_!"«»"]
- convertir tous les textes en miniscules
- transformer les caractères avec accent, chapeau en caractères normaux comme suit â -> a, é -> e, è -> e, û -> u, ...

```
nettoyageTexte = function(text, regExp, remplacement=TRUE) {
  text = str_remove_all(text,regExp)
  lower = str_to_lower(text)
  res = lower
  if (remplacement) {
    res = str_replace_all(lower, c(é = "e", ë = "e", ï = "i", î = "i", à = "a", â = "a", è ="e", ô ="o"
  }
```

```
    return (res)
}
```

Application au texte

```
regExp = "[(),-,.:;-?_!''’«»\"]"
lines = nettoyageTexte(intialTexts, regExp)
```

Création de la matrice de fréquence f

Je crée une `matrice f` qui contient la fréquence des symboles dans un texte.
La `matrice f` est de dimension 90x27, soit **90 lignes(textes) et 27 colonnes(les 26 lettres de l'alphabet et le caractère espace)**

```
createMatriceFreq = function(lines, div = TRUE) {
  nLines = length(lines)
  letters_space<-c(letters," ")
  f = matrix(0, nLines, length(letters_space))
  for (i in 1:nLines) {
    # print(lines[i])
    split = unlist(strsplit(lines[i], split=""))
    res = table(split)
    for (j in 1:length(letters_space)) {
      res2 = res[letters_space[j]]
      if (is.na(as.integer(res2))) {
        f[i,j] = 0
      } else {
        length_Texte = ifelse(div, str_length(lines[i]), 1)
        f[i,j] = as.integer(res2)/length_Texte
      }
    }
  }
  fWithLabels = f
  row.names(fWithLabels)<- 1:nLines
  colnames(fWithLabels)<-letters_space
  return (list(fWithLabels = fWithLabels, f = f))
}

f = createMatriceFreq(lines)$f
print(paste("La dimension de la matrice f est:",toString(dim(f))))
```

```
## [1] "La dimension de la matrice f est: 90, 27"
```

2. Matrice `X` créée à partir de la matrice f

J'utilise la fonction `log` sur les fréquences telles que Xij = log(1+fij) pour harmonizer les données

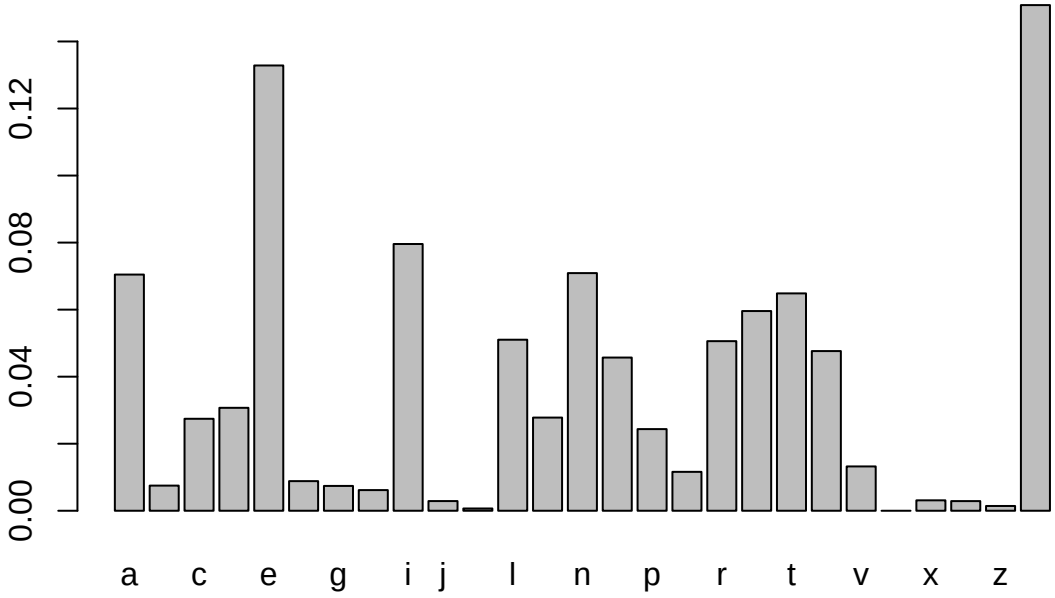
```
createX = function(f, lines) {
  nLines = length(lines)
  newMat = matrix(0, nrow(f), ncol(f))
  newMat = log(1+f)
```

```
  return (newMat)
}

X = createX(f, lines)

# print("La matrice X pour les 10 premiers textes est: ")
# print(X[1:10,1:27])
```

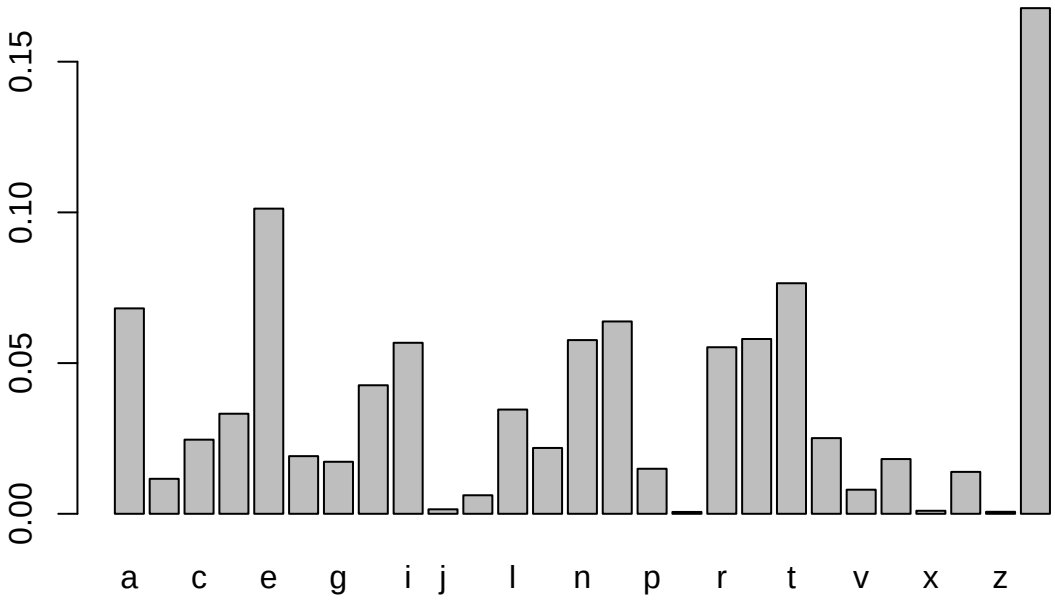
3. Histogramme pour chacune des 2 langues

Dans cette partie, j'affiche les log fréquences des textes en français en anglais et en français

```
showHist = function(X, y, type) {
  indices = which(y == type)
  # print("dans showHist")
  # print(lines[indices])
  letters_space<-c(letters," ")
  tab = rep(0, length(letters_space))
  #print(tab)
  for (j in 1:length(letters_space)) {
    for(i in indices) {
      tab[j] = tab[j] + X[i,j]
    }
  }
  #tab = log(1+tab)
  tmp = tab
  tabNormalized = tab/sum(tab)
  tabNormalizedWithLabels = tabNormalized
  names(tabNormalizedWithLabels) = letters_space
  return (list(tab=tab, tabNormalized = tabNormalized, tabNormalizedWithLabels=tabNormalizedWithLabels))
}


logFreqFrench = showHist(X, y, -1)
barplot(logFreqFrench$tabNormalizedWithLabels, main="Fréquences en français")
```




```
logFreqEnglish = showHist(X, y, 1)
barplot(logFreqEnglish$tabNormalizedWithLabels, main="Fréquences en anglais")
```



Commentaires

Je peux remarquer pour les textes en français, j'ai plus grande fréquence des lettres "e", "j" et "q" qu'en anglais. Parallèlement, j'observe une plus grande fréquence des lettres "k", "y" et "w" qui sont très présence dans la langue anglaise. Ces fréquences sont très logique compte tenu de ces deux langues et de la structure de leurs mots respectifs.

Exercice 2: A pproche gaussian mixture(algorithme EM)

Je vais utiliser **l'algorithme EM** pour déterminer les paramètres de la mixture gaussienne

Initialisation

```
K = 2

p = dim(X)[2]

initParams = function(mat, K) {
  pis = rep(1/K,K)
  nRow = dim(mat)[1]
  nCol = dim(mat)[2]
  rand = sample(1:nRow,K)
  mus = list()
  sigmas = c()
  mus[[1]] = X[1,]
  mus[[2]] = X[1,]
  for (k in 1:K) {
    sigmas[k] = k
  }

  return(list(pis = pis,mus = mus, sigmas = sigmas))
}

initParameters = initParams(X, K)

print(initParameters$pis)
```

```
## [1] 0.5 0.5
```

```
print(initParameters$mus)
```

```
## [[1]]
##  [1] 0.068053463 0.005617992 0.023661507 0.035965149 0.129297088 0.011204599
##  [7] 0.002812941 0.004216450 0.062776406 0.000000000 0.000000000 0.049460630
## [13] 0.016760169 0.054808236 0.052138008 0.026408762 0.008415197 0.066736807
## [19] 0.050800215 0.060127398 0.049460630 0.029148490 0.000000000 0.000000000
## [25] 0.000000000 0.000000000 0.153748184
##
## [[2]]
##  [1] 0.068053463 0.005617992 0.023661507 0.035965149 0.129297088 0.011204599
##  [7] 0.002812941 0.004216450 0.062776406 0.000000000 0.000000000 0.049460630
## [13] 0.016760169 0.054808236 0.052138008 0.026408762 0.008415197 0.066736807
## [19] 0.050800215 0.060127398 0.049460630 0.029148490 0.000000000 0.000000000
## [25] 0.000000000 0.000000000 0.153748184
```

```
# print(diag(rep(initParameters$sigmas[1], p)))
```

Fonction etape E

Ici je calcule la vraisemblance

```
etapeE = function(mat, K, parameters) {
  n = dim(mat)[1]
  p = dim(mat)[2]
  E = matrix(0, n, K)
  for (k in 1:K) {
    tmp = dmvnorm(mat, mean=parameters$mus[[k]],  sigma= diag(rep(parameters$sigmas[k], p)))
    E[,k] = parameters$pis[k]*tmp
  }
  likehood = E
  res =t(apply(E,1,function(x){x/sum(x)}))

  return(list(E= res, likehood = likehood))
}

E = etapeE(X, K, initParameters)$E
```

Fonction etape M

Dans cette partie, je cherche les paramètres qui maximisent la vraisemblance

```
etapeM = function(mat, E, K) {
  pis = rep(0,K)
  mus = list()
  sigmas = c()
  n = dim(mat)[1]
  data_ssq <- apply(mat^2, 1, sum)
  p = ncol(mat)
  mu_sq = c()
  for(k in 1:K) {
    Nk = sum(E[,k])
    mus[[k]] = apply(E[,k]*mat, 2, sum)/Nk
    tmp = t(mat) - mus[[k]]
    tmp2 = t(tmp)
    mu_sq[k] <- sum(mus[[k]]^2) / p
    sigmas[k] = sum(E[,k]*data_ssq/(p*Nk)) - mu_sq[k]
        pis[k] = Nk/n
  }
  return(list(pis = pis,mus = mus, sigmas = sigmas))
}
```

Fonction EM combinant les étapes d'initialisation, E et M

J'effectue les trois étapes jusqu'à convergence. Notre test se fait sur le logLikehood, je regarde la différence du logLike du précédent calcul et l'actuel par rapport à tol = 10^-6

```
EM = function(mat, K) {
  parameters<-initParams(mat,K)
  nRow = dim(mat)[1]
  E<-matrix(1,nRow,K)
  logLikehood = c(0,1)
  i = 2
  tol = 10^-6
```

```
  while(abs(logLikehood[i]-logLikehood[i-1]) >= tol){
    old.parameters<-parameters
    i = i+1
    E<-etapeE(mat,K,parameters)$E
    parameters<-etapeM(mat,E, K)
    # delta = sum(((parameters$mus[[1]] - old.parameters$mus[[1]]))^2)
    likehood = etapeE(mat,K,parameters)$likehood

    obsloglik <- sum(apply(likehood,1, function(x) {log(sum(x))}))
    logLikehood[i] = obsloglik
  }
  return(list(E=E,parameters=parameters, logLikehood = logLikehood))
}

EM_all_data = EM(X, K)

print("les paramètres obtenus à la fin")
```

```
## [1] "les paramètres obtenus à la fin"
```

```
cat("\n")
```

```
print("Paramètres pis")
```

```
## [1] "Paramètres pis"
```

```
print(EM_all_data$parameters$pis)
```

```
## [1] 0.5000041 0.4999959
```

```
cat("\n")
```

```
print("Paramètres mus")
```

```
## [1] "Paramètres mus"
```

```
print(EM_all_data$parameters$mus)
```

```
## [[1]]
##  [1] 6.773658e-02 7.208939e-03 2.637746e-02 2.952406e-02 1.277018e-01
##  [6] 8.499114e-03 7.118445e-03 5.929568e-03 7.650564e-02 2.776398e-03
## [11] 6.518184e-04 4.905232e-02 2.672499e-02 6.816044e-02 4.394103e-02
## [16] 2.341422e-02 1.115922e-02 4.863986e-02 5.727667e-02 6.234049e-02
## [21] 4.579237e-02 1.271030e-02 1.135435e-07 2.992540e-03 2.766097e-03
## [26] 1.367333e-03 1.450228e-01
##
## [[2]]
##  [1] 0.0656188439 0.0111785338 0.0236840096 0.0319678944 0.0974829495
##  [6] 0.0184174162 0.0166189076 0.0410619003 0.0546325193 0.0014347332
## [11] 0.0059181163 0.0333030148 0.0210320260 0.0554866063 0.0614479782
## [16] 0.0143885065 0.0005909881 0.0532088177 0.0558428307 0.0736522695
## [21] 0.0241762970 0.0076991127 0.0174887514 0.0009628781 0.0133887398
## [26] 0.0006362378 0.1615322445
```

```
# print(diag(rep(EM_all_data$parameters$sigmas[1],p)))
# print(diag(rep(EM_all_data$parameters$sigmas[2],p)))

print("les résultats du loglike durant l'itération")
```

```
## [1] "les résultats du loglike durant l'itération"
```

```
print(EM_all_data$logLikehood)
```

```
##  [1]    0.000    1.000 7680.831 7680.831 7680.836 7735.876 8118.387 8304.240
##  [9] 8354.864 8356.015 8356.015 8356.015
```

3. Performance du classifieur

Cette partie est en deux étapes, j'évalue la performance du classifieur:

1. sur toutes les données
2. par validation

1. Sur toutes les données

```
ConfusionMatrix = function(y, y_pred) {
  mat = table(y, y_pred)
  return (mat)
}

# je transforme les 1 en -1 et les 2 en 1 de nos prédictions

transformTab = function(tab) {
  for (i in 1:length(tab)) {
    if (tab[i] == 1) {
      tab[i] = -1
    } else {
      tab[i] = 1
    }
  }
  return (tab)
}

# je prédit la classe en prenant le max entre les 2 colonnes pour chaque ligne
getClass = function(E) {
  tmp = apply(E, 1, which.max)
  res = transformTab(tmp)
  val = list(transformed = res, noTransformed = tmp)
  return (val)
}

parameters_all_data = EM_all_data$parameters

predict_proba_all_data = etapeE(X, K, parameters_all_data)$E

predict_class_all_data = getClass(predict_proba_all_data)$transformed
matDeConfusion_all_data = ConfusionMatrix(y, predict_class_all_data)
print("Matrice de confusion sur toutes les données est: ")
```

```
## [1] "Matrice de confusion sur toutes les données est: "
```

```
print(matDeConfusion_all_data)
```

```
##     y_pred
## y    -1  1
##   -1 45  0
##   1   0 45
```

2. Validation croisée

Grâce à la fonction `createDataPartition` de `caret`, j'effectue une validation croisée sur nos données. On procède comme suit:

- Je fais les données d'entraînement et de test de façon aléatoire avec `createDataPartition` de la librairie `caret`

- J'évalue les paramètres sur les 30 derniers textes et je evalue ses performances sur les 30 premiers textes
- Je réalise un kfold sur 3 groupes de textes avec les paramètres de 60 textes comme données d'entraînement et 30 textes comme test

Validation croisée avec createDataPartition

Le but est de capturer le maximum de différence entre les différents textes en espérant choisir les données d'entraînement qui reperésentent le mieux les données

```
  trainIndex<- createDataPartition(y, p = 0.5, list=FALSE)
  length(trainIndex)
```

```
## [1] 46
```

```
  X_train = X[trainIndex,]
  y_train = y[trainIndex]
  X_test = X[-trainIndex,]
  y_test = y[-trainIndex]

  EM_cv = EM(X_train, 2)
  parameters_cv = EM_cv$parameters
  predict_proba_cv = etapeE(X_test, 2, parameters_cv)$E
  predict_class_cv = getClass(predict_proba_cv)$transformed
  matDeConfusion_cv = ConfusionMatrix(y_test, predict_class_cv)

  print("Matrice de confusion par validation croisée: ")
```

```
## [1] "Matrice de confusion par validation croisée: "
```

```
  print(matDeConfusion_cv)
```

```
##     y_pred
## y    -1  1
##   -1 22  0
##   1   0 22
```

Validation croisée sur les 30 premiers et test sur les 30 autres restants

Dans cette partie, j'ai les paramètres sur les 30 premiers textes etj'aimerait évaluer ses performances sur les 30 restants

```
  trainIndex2 = 1:30
  X_train2 = X[trainIndex2,]
  y_train2 = y[trainIndex2]
  X_test2 = X[-trainIndex2,]
  y_test2 = y[-trainIndex2]

  EM_cv2 = EM(X_train2, 2)
  parameters_cv2 = EM_cv2$parameters
  predict_proba_cv2 = etapeE(X_test2, 2, parameters_cv2)$E
  predict_class_cv2 = getClass(predict_proba_cv2)$transformed
  matDeConfusion_cv2 = ConfusionMatrix(y_test2, predict_class_cv2)

  print("Matrice de confusion par validation croisée (train = 30 textes, test = 30 textes) : ")
```

```
## [1] "Matrice de confusion par validation croisée (train = 30 textes, test = 30 textes) : "
```

```
  print(matDeConfusion_cv2)
```

```
##     y_pred
## y    -1  1
##   -1 30  0
##   1   0 30
```

Les données sont bien melangées au début, ce qui fait qu'on a des données d'entraînement qui sont représentatives des données de test.

Validation croisée sur chaque groupe de 30 textes entrainés sur les 60 autres

Ici Je réalise un kfold sur les données.

```
Kf=3
size=30
Acc=rep(NA,Kf)
for(k in 1:Kf){
  k1=k*size-(size-1)
  k2=k*size
  Xtrain=X[-(k1:k2),]
  Xtest=X[k1:k2,]

  EMtrain=EM(Xtrain,2)
  parms=EMtrain$parameters

  Etest=etapeE(Xtest,2,parms)$E
  y_test = getClass(Etest)$transformed
  #y_test1 = getClass(Etest)$noTransformed

  matDeConfusion = ConfusionMatrix(y[k1:k2], y_test)

  print("matrice de confusion")
  print(matDeConfusion)
  # print("y")
  # print(y[k1:k2])
  # print("y_test")
  # print(y_test)
  Acc[k]=(matDeConfusion[1,1]+matDeConfusion[2,2])/sum(matDeConfusion)
```

```
}
```

```
## [1] "matrice de confusion"
##     y_pred
## y    -1  1
##   -1 15  0
##   1   0 15
## [1] "matrice de confusion"
##     y_pred
## y    -1  1
##   -1 12  0
##   1   0 18
## [1] "matrice de confusion"
##     y_pred
## y    -1  1
##   -1 18  0
##   1   0 12
```

```
cat("\n")
```

```
print("Précision du classifieur de Bayes")
```

```
## [1] "Précision du classifieur de Bayes"
```

```
print(mean(Acc))
```

```
## [1] 1
```

On voit la performance du classifieur sur nos données.

Exercice 3: Approche par chaîne de markov caché

1. Estimation des paramètres

Dans cette partie je crée un classifieur markovien basé sur les matrices de transitions entre nos différents symboles des différentes.

Fonctions utiles pour estimer les paramètres

```
### Fonction qui renvoie les textes d'une langue
getTextByLanguage = function(lines, y, type) {
  indices = which(y == type)
  res = lines[indices]
  return (res)
}


### Matrice de transition
transitionMatrix = function(bigrams){
  letters_space<-c(letters," ")
  newMat<-matrix(0,27,27)
  row.names(newMat)<-letters_space
  colnames(newMat)<-letters_space
```

```
  for (letteri in letters_space)
    for (letterj in letters_space)
      if ((letteri %in% row.names(bigrams))&&(letterj %in% row.names(bigrams))) newMat[letteri,letterj]
  return (newMat)
}

## Fonction de normalisation afin que la somme des lignes soit égale 1

normaliseMatrix = function(mat) {
  n = dim(mat)[1]
  p = dim(mat)[2]
  newMat = matrix(0, n, p)

  ###########j'additionne à notre matrice initiale pour éviter le problème de division par zéros

  mat = mat + 1
  for(i in 1:n) {
    newMat[i,] = mat[i,]/sum(mat[i,])
  }
  return (newMat)
}

# Calcul de la proba initiale
probaInitiale = function(A) {
  n = dim(A)[2]
  M<-diag(rep(1,n))-A
  M[,n]<-rep(1,n)
  b = rep(0, n)
  b[n] = 1
  Pi<-solve(t(M),b=b)
  return (Pi)
}


# Fonction  Simulation de chaîne de Markov par classe c'est à dire par langue
## lines nos textes
## y les classes de nos textes
# La classe pour laquelle je estime les paramètres y = -1 ou  y = 1

estimMarkovParameters = function(lines, y, type) {

  # je détermine dans y les indcices où y = type
  indices = which(y == type)

  # je récupère les textes correspondants aux indices
  texte = lines[indices]

  # opération de traitement pour créer un bigramme
  textParLangue1 = c(" ",texte)
  tmp1 = paste(textParLangue1, collapse="")
  textParLangue1 = unlist(strsplit(tmp1, split=""))
  textParLangue2 = c(texte," ")
  tmp2 = paste(textParLangue2, collapse="")
```

```
  textParLangue2 = unlist(strsplit(tmp2, split=""))

  bigrams<-table(textParLangue2,textParLangue1)

  # je normalise la matrice en ajoutant 1 à chaque élément(pour éviter la division par zéro) de la lign
  # je divise chaque élément par la somme des éléments
  A = normaliseMatrix(transitionMatrix(bigrams))

  Pi = probaInitiale(A)

  parameters = list(A = A, Pi = Pi)
  return (parameters)
}
```

Estimation des paramètres

```
parameters1 = estimMarkovParameters(lines, y, -1)
parameters2 = estimMarkovParameters(lines, y, 1)

# print("Les paramètres pour le français sont: ")
# print(parameters1)
#
# print("Les paramètres pour l'anglais sont: ")
# print(parameters2)
#
# print("somme des colonnes de pi pour le français")
# print(sum(parameters1$Pi))
#
# print("somme des lignes de A pour le français")
# print(apply(parameters1$A,1,sum))
#
# print("somme des colonnes de pi pour l'anglais")
# print(sum(parameters2$Pi))
#
# print("somme des lignes de A pour l'anglais")
# print(apply(parameters2$A,1,sum))

B1 = parameters1$Pi
B2 = parameters2$Pi
B = matrix(c(B1, B2), 27,2)
```

2. Programmation du classifieur

Je simule d'abord les deux chaînes de markov à partir des paramètres calculés précédemment.

A partir de notre probabilité d'émission une matrice de dimension 30x2, je prédis la classe des textes.

```
# Fonction pour simuler une chaîne de Markov

simulerCM = function(n, A, Pi) {
  Z<-rep(0,n);
  nbRow = nrow(A)
```

```
  Z[1]<-sample(1:nbRow,prob=Pi,size=1)
  for (i in 2:n) {
    Z[i] = sample(1:nbRow,prob=A[Z[i-1],],size=1)
  }
  return (Z)
}

#j'ai deux chaînes de markov simulées suivant les paramètres A1, Pi1, A2, Pi2

CM1 = simulerCM(dim(X)[1], parameters1$A, parameters1$Pi)
CM2 = simulerCM(dim(X)[1], parameters2$A, parameters2$Pi)

classifMarkov = matrix(0, dim(X)[1],K)
classifMarkov[,1] = B[CM1,1]
classifMarkov[,2] = B[CM2,2]
predict_class_markov = getClass(classifMarkov)$transformed
# print(predict_class_markov)
matDeConfusion_markov = ConfusionMatrix(y, predict_class_markov)
print("Matrice de confusion du classifieur markovien")
```

```
## [1] "Matrice de confusion du classifieur markovien"
```

```
print(matDeConfusion_markov)
```

```
##     y_pred
## y    -1  1
##   -1 29 16
##   1  23 22
```

Validation croisée avec createDataPartition

Le but est de capturer le maximum de différence entre les différents textes en espérant choisir les données d'entraînement qui reperésentent le mieux les données

```
  trainIndex_markov<- createDataPartition(y, p = 0.5, list=FALSE)
  length(trainIndex_markov)
```

```
## [1] 46
```

```
  X_train_markov = lines[trainIndex_markov]
  y_train_markov = y[trainIndex_markov]
  X_test_markov = lines[-trainIndex_markov]
  y_test_markov = y[-trainIndex_markov]

  parameters1_train_cv = estimMarkovParameters(X_train_markov, y_train_markov, -1)
  parameters2_train_cv = estimMarkovParameters(X_train_markov, y_train_markov, 1)
  B_train_cv = matrix(c(parameters1_train_cv$Pi, parameters2_train_cv$Pi), 27,2)

  parameters1_test_cv = estimMarkovParameters(X_test_markov, y_test_markov, -1)
  parameters2_test_cv = estimMarkovParameters(X_test_markov, y_test_markov, 1)

  CM1_cv = simulerCM(dim(X)[1], parameters1_test_cv$A, parameters1_test_cv$Pi)
  CM2_cv = simulerCM(dim(X)[1], parameters2_test_cv$A, parameters2_test_cv$Pi)

classifMarkov_cv = matrix(0, dim(X)[1],K)
```

```
classifMarkov_cv[,1] = B_train_cv[CM1_cv,1]
classifMarkov_cv[,2] = B_train_cv[CM2_cv,2]
predict_class_markov_cv = getClass(classifMarkov_cv)$transformed
# print(predict_class_markov)
matDeConfusion_markov_cv = ConfusionMatrix(y, predict_class_markov_cv)
print("Matrice de confusion du classifieur markovien")
```

```
## [1] "Matrice de confusion du classifieur markovien"
```

```
print(matDeConfusion_markov_cv)
```

```
##     y_pred
## y    -1  1
##   -1 21 24
##   1  21 24
```

Exercice 3: Viterbi

1. Création d'un court texte

Le court texte est dans le fichier viterbi.csv Chaque est une observation dont va essayer de déterminer la langue

```
# file = "/home/boua/Documents/Tps/2020_2021/Non_supervisé/Projet/viterbi.csv"
# test = read.csv(file, sep = "|")
# # lines <- readLines()
# court_texte = as.vector(test$Textes)


phrase1 = "C'est ainsi qu'il restructure la destructuration circonstancielle du nativisme"
phrase2 = "this is the price and the promise of citizenship  of America's the coldest country"
phrase3 = "cependant mitiger ce raisonnement car il envisage l'expression générative du nativisme"
phrase4 = "the year of America's birth, in the coldest of month"
phrase5 = "Mais il ne faut pas oublier pour autant qu'il envisage la destructuration empirique du nativ
phrase6 = "Homes have been lost; jobs shed; businesses shuttered"
court_texte = paste(c(phrase1, phrase2, phrase3,  phrase4, phrase5, phrase6), collapse = " ")

print(paste("La taille du court texte  est : ", toString(sum(str_length(court_texte)))))
```

```
## [1] "La taille du court texte  est :  452"
```

```
cat("\n")
```

```
print("Le court texte est: ")
```

```
## [1] "Le court texte est: "
```

```
print(court_texte)
```

```
## [1] "C'est ainsi qu'il restructure la destructuration circonstancielle du nativisme this is the pric
```

2. Application du viterbi

Les paramètres à passer à l'algorithme de viterbi.

La matrice de transition **A_viterbi** est faite en se basant sur l'alternance des langues des différentes phrases.
Le poids **Pi** est obtenu en appliquant la formule qui permet de calculer la probabilité initiale à partir de la matrice de transition.
La probabilité **B** d'émission est la somme des deux **Pi** calculés.

```
K = 2
# Matrice de transition

# A = matrix(c(0.1,0.9,0.9,0.1),K,K)
A_viterbi = matrix(c(0.9,0.1,0.1,0.9),2,2)

print("Matrice de transition pour l'algorithme de viterbi")
```

```
## [1] "Matrice de transition pour l'algorithme de viterbi"
```

```
print(A_viterbi)
```

```
##      [,1] [,2]
## [1,]  0.9  0.1
## [2,]  0.1  0.9
```

```
cat("\n")
```

```
# Calcul de la probabilité initiale
M<-diag(rep(1,K))-A_viterbi
M[,K]<-rep(1,K)
Pi_viterbi<-solve(t(M),b=c(0,1))
print("Probabilité initiale obtenue après calcul")
```

```
## [1] "Probabilité initiale obtenue après calcul"
```

```
print(Pi_viterbi)
```

```
## [1] 0.5 0.5
```

```
cat("\n")
```

```
# Probabilité d'émission calculée dans le classifieur de Markov à partir de A1 et A2 les matrices de tr
B1 = parameters1$Pi
B2 = parameters2$Pi
B = matrix(c(B1, B2), 27,2)
print("Probabilité d'émission")
```

```
## [1] "Probabilité d'émission"
```

```
print(B)
```

```
##               [,1]        [,2]
##  [1,] 0.0701088631 0.067537251
##  [2,] 0.0082857617 0.011894231
##  [3,] 0.0265175385 0.024783770
##  [4,] 0.0307789876 0.032781541
##  [5,] 0.1348591796 0.100455527
##  [6,] 0.0095056477 0.018709050
##  [7,] 0.0079027805 0.017502954
```

```
##  [8,] 0.0069614701 0.041563737
##  [9,] 0.0759612445 0.057410922
## [10,] 0.0039692426 0.002380207
## [11,] 0.0014623319 0.006755859
## [12,] 0.0498895175 0.035137005
## [13,] 0.0268841771 0.022810443
## [14,] 0.0680206293 0.056685276
## [15,] 0.0441508596 0.062004455
## [16,] 0.0234959671 0.015613805
## [17,] 0.0118153039 0.001505495
## [18,] 0.0506478862 0.053521517
## [19,] 0.0577244565 0.056895961
## [20,] 0.0631967648 0.074841808
## [21,] 0.0471609327 0.024222413
## [22,] 0.0134748598 0.008607712
## [23,] 0.0009401199 0.017826936
## [24,] 0.0040051191 0.001995352
## [25,] 0.0034814040 0.014083488
## [26,] 0.0020543797 0.001645668
## [27,] 0.1567445750 0.170827618
```

```
cat("\n")
```

```
print("Somme des deux colones de B")
```

```
## [1] "Somme des deux colones de B"
```

```
print(apply(B, 2, sum))
```

```
## [1] 1 1
```

Quelques fonctions utiles pour l'affichage des résultats et pour le calcul dans l'algorithme de viterbi

```
# Fonction pour l'affichage des différents passages selon la langue à partir des séquences de classes o
get_language = function(tab, type) {
  incr = 1
  listIndex = list()
  start =
  listIndex[[incr]] = c(which(tab == type)[1])
  for(i in 2:length(tab)) {
    if(tab[i] != tab[i-1]) {
      if (tab[i-1] == type) {
        listIndex[[incr]] = c(listIndex[[incr]], i-1)
      }
      else {
          if (is.null(listIndex[[1]]) || length(listIndex[[1]]) > 1) {
            incr = incr + 1
          }
          listIndex[[incr]] = c(i)
      }
    }
  }

  # listIndex[[incr]] = ifelse(c(listIndex[[incr]], length(tab)))
```

```
  if (length(listIndex[[incr]]) == 1) {
    listIndex[[incr]] = c(listIndex[[incr]], length(tab))
  }
  return (listIndex)
}

# Calcule la log fréquence à partir de la matrice B
calculateSeq = function(seq, B, k) {
  splits = unlist(strsplit(seq, split=""))
  tmp = 0
  lettersWithSpace = c(letters, " ")
  for (split in splits ) {
    ind = which(lettersWithSpace == split)
    tmp<- tmp + log(B[ind,k])
  }
  return (tmp)
}
```

code pour l'algorithme de viterbi

```
Viterbi<-function(Pi,A,B,observations){
  K<-nrow(A)
  T<-length(observations)
  S<-matrix(0,K,T)
  logV<-matrix(-Inf,K,T)
  Zest<-rep(0,T)
  for (k in 1:K){
    logV[k,1] = calculateSeq(observations[1], B, k) + log(Pi[k])
    S[k,1]<-0

  }

  # Forward
  for (t in (2:T))
    for (k in (1:K)){
      probSeq = calculateSeq(observations[t], B, k)
      logV[k,t]=max(logV[,t-1]+log(A[,k])+probSeq)
      S[k,t-1]=which.max(logV[,t-1]+log(A[,k])+probSeq)
    }

  # Backward
  Zest[T]<-which.max(logV[,T])
  for (t in (T-1):1)
    Zest[t]<-S[Zest[t+1],t]
  return(Zest)
}


contenateAllTextes = function(intialTexts) {
  regExp = "[(),,-.:';-?_!«»\"]"
  tmp = nettoyageTexte(intialTexts, regExp)
  # res = paste(tmp, collapse="")
```

```
  res = tmp
  return (res)
}

observation_tmp = contenateAllTextes(court_texte)
observation = unlist(strsplit(observation_tmp, split=""))

print("Observations à passer dans l'algorithme de viterbi")
```

```
## [1] "Observations à passer dans l'algorithme de viterbi"
```

```
print(observation)
```

```
##   [1] "c" "e" "s" "t" " " "a" "i" "n" "s" "i" " " "q" "u" "i" "l" " " "r" "e"
##  [19] "s" "t" "r" "u" "c" "t" "u" "r" "e" " " "l" "a" " " "d" "e" "s" "t" "r"
##  [37] "u" "c" "t" "u" "r" "a" "t" "i" "o" "n" " " "c" "i" "r" "c" "o" "n" "s"
##  [55] "t" "a" "n" "c" "i" "e" "l" "l" "e" " " "d" "u" " " "n" "a" "t" "i" "v"
##  [73] "i" "s" "m" "e" " " "t" "h" "i" "s" " " "i" "s" " " "t" "h" "e" " " "p"
##  [91] "r" "i" "c" "e" " " "a" "n" "d" " " "t" "h" "e" " " "p" "r" "o" "m" "i"
## [109] "s" "e" " " "o" "f" " " "c" "i" "t" "i" "z" "e" "n" "s" "h" "i" "p" " "
## [127] " " "o" "f" " " "a" "m" "e" "r" "i" "c" "a" "s" " " "t" "h" "e" " " "c"
## [145] "o" "l" "d" "e" "s" "t" " " "c" "o" "u" "n" "t" "r" "y" " " "c" "e" "p"
## [163] "e" "n" "d" "a" "n" "t" " " "m" "i" "t" "i" "g" "e" "r" " " "c" "e" " "
## [181] "r" "a" "i" "s" "o" "n" "n" "e" "m" "e" "n" "t" " " "c" "a" "r" " " "i"
## [199] "l" " " "e" "n" "v" "i" "s" "a" "g" "e" " " "l" "e" "x" "p" "r" "e" "s"
## [217] "s" "i" "o" "n" " " "g" "e" "n" "e" "r" "a" "t" "i" "v" "e" " " "d" "u"
## [235] " " "n" "a" "t" "i" "v" "i" "s" "m" "e" " " "t" "h" "e" " " "y" "e" "a"
## [253] "r" " " "o" "f" " " "a" "m" "e" "r" "i" "c" "a" "s" " " "b" "i" "r" "t"
## [271] "h" " " "i" "n" " " "t" "h" "e" " " "c" "o" "l" "d" "e" "s" "t" " " "o"
## [289] "f" " " "m" "o" "n" "t" "h" " " "m" "a" "i" "s" " " "i" "l" " " "n" "e"
## [307] " " "f" "a" "u" "t" " " "p" "a" "s" " " "o" "u" "b" "l" "i" "e" "r" " "
## [325] "p" "o" "u" "r" " " "a" "u" "t" "a" "n" "t" " " "q" "u" "i" "l" " " "e"
## [343] "n" "v" "i" "s" "a" "g" "e" " " "l" "a" " " "d" "e" "s" "t" "r" "u" "c"
## [361] "t" "u" "r" "a" "t" "i" "o" "n" " " "e" "m" "p" "i" "r" "i" "q" "u" "e"
## [379] " " "d" "u" " " "n" "a" "t" "i" "v" "i" "s" "m" "e" " " "h" "o" "m" "e"
## [397] "s" " " "h" "a" "v" "e" " " "b" "e" "e" "n" " " "l" "o" "s" "t" " " "j"
## [415] "o" "b" "s" " " "s" "h" "e" "d" " " "b" "u" "s" "i" "n" "e" "s" "s" "e"
## [433] "s" " " "s" "h" "u" "t" "t" "e" "r" "e" "d"
```

```
res_viterbi = Viterbi(Pi_viterbi, A_viterbi,B, observation)
```

Affichage des résultas donnés par l'algorithme de viterbi

```
print("La sortie de l'algorithme de Viterbi")
```

```
## [1] "La sortie de l'algorithme de Viterbi"
```

```
print(res_viterbi)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [112] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [149] 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [260] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [297] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [334] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [371] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [408] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```
cat("\n")
```

```
# je transforme les 1 en -1 et les 2 en 1
# print("on transforme les classes prédites, les 1 en -1 et les 2 en 1")
# viterbi_class_transformed = transformTab(res_viterbi)
# print(viterbi_class_transformed)
# cat("\n")

## Affichage du résultat
print("****************************************** Affichage des passages detectés par viterbi ******
```

```
## [1] "****************************************** Affichage des passages detectés par viterbi *****
```

```
cat("\n")
```

```
indicesFrench = get_language(res_viterbi, 1)
indicesEnglish = get_language(res_viterbi, 2)
cat("\n")
```

```
print("**************************Les différents passages en français detectés par viterbi************
```

```
## [1] "**************************Les différents passages en français detectés par viterbi************
```

```
cat("\n")
```

```
for (i in 1:length(indicesFrench)) {
  aAfficher1 = paste("Du caractère ", toString(indicesFrench[[i]][1])," à ", toString(indicesFrench[[i]
  aAfficher2 = paste(observation[indicesFrench[[i]][1]:indicesFrench[[i]][2]], collapse = "")
  print(aAfficher1)
  print(aAfficher2)
  cat("\n")
}
```

```
## [1] "Du caractère  1  à  76"
## [1] "cest ainsi quil restructure la destructuration circonstancielle du nativisme"
##
## [1] "Du caractère  160  à  244"
## [1] "cependant mitiger ce raisonnement car il envisage lexpression generative du nativisme"
##
## [1] "Du caractère  297  à  391"
## [1] "mais il ne faut pas oublier pour autant quil envisage la destructuration empirique du nativisme
```

```
cat("\n")
```

```
print("**************************Les différents passages en anglais detectés par viterbi**************
```

```
## [1] "**************************Les différents passages en anglais detectés par viterbi************
```

```
cat("\n")
```

```
for (i in 1:length(indicesEnglish)) {
  aAfficher1 = paste("Du caractère ", toString(indicesEnglish[[i]][1])," à ", toString(indicesEnglish[[
  aAfficher2 = paste(observation[indicesEnglish[[i]][1]:indicesEnglish[[i]][2]], collapse = "")
```

```
  print(aAfficher1)
  print(aAfficher2)
  cat("\n")
}
```

```
## [1] "Du caractère  77  à  159"
## [1] " this is the price and the promise of citizenship  of americas the coldest country "
##
## [1] "Du caractère  245  à  296"
## [1] " the year of americas birth in the coldest of month "
##
## [1] "Du caractère  392  à  443"
## [1] " homes have been lost jobs shed businesses shuttered"
```

Commentaires

Je remarque que dans cette méthode, à chaque étape la probabilité d'émission du caractère et du caractère précédent joue un rôle important dans la détection des passages et des couples de caractères(bigrammes)

Exercice4: algorithme de Baum-Welch

1. Algorithme de Baum-Welch pour deux états cachés

Dans cette partie, je code les fonctions `forward`, `backward` ainsi que les méthodes d'estimation de la `matrice de transition`` et des probabilités d'émission` pour deux états cachés.
Le plus gros problème lors de la modélisation d'un HMM étant de l'imprécision des flottants, J'utiliserai dans les fonctions `forward`, `backward` des méthodes pour éviter de tendre vers l'infini ou être à zéro.

Forward

```
forward = function(hmm, observation)
{
  hmm$A[is.na(hmm$A)]       = 0
  hmm$B[is.na(hmm$B)] = 0
  nObservations  = length(observation)
  nStates    = nrow(hmm$A)
  f          = array(NA,c(nStates,nObservations))
  # Init
  for(state in 1:nStates)
  {
    probSeq = calculateSeq(observation[1], hmm$B, state)
    f[state,1] = log(hmm$Pi[state]) + probSeq
  }


  # Iteration
  for(k in 2:nObservations)
  {
    for(state in 1:nStates)
```

```
    {
      logsum = -Inf
      for(previousState in 1:nStates)
      {
        temp   = f[previousState,k-1] + log(hmm$A[previousState,state])
          if(temp > - Inf)
          {
            logsum = temp + log(1 + exp(logsum - temp ))
          }
      }
      probSeq = calculateSeq(observation[k], hmm$B, state)
      f[state,k] = probSeq + logsum
    }
  }
  return(f)
}
```

Backward

```
backward = function(hmm, observation)
{
  lettersWithSpace = c(letters, " ")
  hmm$A[is.na(hmm$A)]       = 0
  hmm$B[is.na(hmm$B)] = 0
  nObservations  = length(observation)
  nStates    = nrow(hmm$A)
  b          = array(NA,c(nStates,nObservations))
  # Init
  for(state in 1:nStates)
  {
    b[state,nObservations] = log(1)
  }
  # Iteration
  for(k in (nObservations-1):1)
  {
    for(state in 1:nStates)
    {
      logsum = -Inf
      for(nextState in 1:nStates)
      {
        ind = which(lettersWithSpace == observation[k+1])
        tmp<- hmm$B[ind,nextState]
        temp   = b[nextState,k+1] + log(hmm$A[state,nextState]*tmp)
        if(temp > - Inf)
        {
          logsum = temp + log(1 + exp(logsum-temp))
        }
      }
      b[state,k] = logsum
    }
  }
  return(b)
```

```
}
```

Etape Calcul de la matrice A et de la probabilité d'émission

Ici j'estime ou réestime la `matrice de transion` et les `probabilités d'émission`

```
baumWelchEstim = function(hmm, observation)
{
  TransitionMatrix    = hmm$A
  TransitionMatrix[,] = 0
  EmissionMatrix      = hmm$B
  EmissionMatrix[,]   = 0
  f = forward(hmm,  observation)
  b = backward(hmm, observation)
  nStates = nrow(hmm$A)
  probObservations = f[1,length(observation)]
  lettersWithSpace = c(letters, ' ')
  for(i in 2:nStates)
  {
    j = f[i,length(observation)]
    if(j > - Inf)
    {
      probObservations = j + log(1+exp(probObservations-j))
    }
  }
  for(x in 1:nStates)
  {
    for(y in 1:nStates)
    {
      probSeqTemp = calculateSeq(observation[1+1], hmm$B, y)
      temp = f[x,1] + log(hmm$A[x,y]) + probSeqTemp + b[y,1+1]

      for(i in 2:(length(observation)-1))
      {
          probSeq = calculateSeq(observation[i+1], hmm$B, y)
          j = f[x,i] + log(hmm$A[x,y]) + probSeq + b[y,i+1]
          if(j > - Inf)
          {
            temp = j + log(1+exp(temp-j))
          }
      }
      temp = exp(temp - probObservations)
      TransitionMatrix[y,x] = temp
    }
  }

  for(x in 1:nStates)
  {
    for(s in hmm$Symbols)
    {
      temp = -Inf
      for(i in 1:length(observation))
      {
```

```
        if(s == observation[i])
        {
          j = (f[x,i] + b[x,i])
          if(j > - Inf)
          {
            temp = j + log(1+exp(temp-j))
          }
        }
      }

      temp = exp(temp - probObservations)
      ind  = which(lettersWithSpace == s)
      EmissionMatrix[ind,x] = temp
    }
  }
  return(list(TransitionMatrix=TransitionMatrix,EmissionMatrix=EmissionMatrix))
}


# BWR = baumWelchEstim(hmm, observation)
```

je regroupe tout ici

```
baumWelch = function(hmm, observation, maxIterations=100, delta=1E-9, pseudoCount=0)
{
  tempHmm = hmm
  #print(observation)
  tempHmm$transProbs[is.na(hmm$transProbs)]       = 0
  tempHmm$emissionProbs[is.na(hmm$emissionProbs)] = 0
  diff = c()
  for(i in 1:maxIterations)
  {
    # Expectation Step (Calculate expected Transitions and Emissions)
    bw = baumWelchEstim(tempHmm, observation)
    T  = bw$TransitionMatrix
    E  = bw$EmissionMatrix
    # Pseudocounts
    T[!is.na(hmm$transProbs)]    = T[!is.na(hmm$transProbs)]    + pseudoCount
    E[!is.na(hmm$emissionProbs)] = E[!is.na(hmm$emissionProbs)] + pseudoCount
    # Maximization Step (Maximise Log-Likelihood for Transitions and Emissions-Probabilities)
    T = (T/apply(T,1,sum))
    E = t(t(E)/apply(E,2,sum))
    d = sqrt(sum((tempHmm$transProbs-T)^2)) + sqrt(sum((tempHmm$emissionProbs-E)^2))
    diff = c(diff, d)
    tempHmm$transProbs    = T
    tempHmm$emissionProbs = E
    if(d < delta)
    {
      break
    }
  }
  tempHmm$transProbs[is.na(hmm$transProbs)]       = NA
```

```
  tempHmm$emissionProbs[is.na(hmm$emissionProbs)] = NA
  return(list(hmm=tempHmm,difference=diff))
}
```

2. Excécution de l'algorithme de Baum-Welch avec comme initialisation les informations de l'exercice précédent

Dans cette partie, j'excécute **l'algorithme de Baum-Welch** en utilisant comme initialisation les informations de l'exercice précédent.

```
lettersWithSpace = c(letters, " ")
hmm = list(A = A_viterbi, B = B, Pi = Pi_viterbi, Symbols = lettersWithSpace)
BW = baumWelch(hmm, observation)

A_baum_welch = BW$hmm$transProbs
B_baum_welch = BW$hmm$emissionProbs

print("Matrice A estimée: ")
```

```
## [1] "Matrice A estimée: "
```

```
print(A_baum_welch)
```

```
##           [,1]       [,2]
## [1,] 0.9110396 0.08896037
## [2,] 0.1076713 0.89232869
```

```
cat("\n")
```

```
print("Probabilité d'émission estimée")
```

```
## [1] "Probabilité d'émission estimée"
```

```
print(BW$hmm$emissionProbs)
```

```
##              [,1]         [,2]
##  [1,] 0.064918590 0.0561630807
##  [2,] 0.009029229 0.0140072181
##  [3,] 0.038691330 0.0379931574
##  [4,] 0.025864068 0.0235853505
##  [5,] 0.117816266 0.1118797926
##  [6,] 0.006400135 0.0171756332
##  [7,] 0.008910315 0.0091727926
##  [8,] 0.008756564 0.0541576760
##  [9,] 0.099642223 0.0740491871
## [10,] 0.001605829 0.0030424904
## [11,] 0.000000000 0.0000000000
## [12,] 0.036818195 0.0203396082
## [13,] 0.024352016 0.0303853112
## [14,] 0.065730841 0.0502064747
## [15,] 0.033491102 0.0542156017
## [16,] 0.020101805 0.0155964615
## [17,] 0.011964792 0.0005140016
## [18,] 0.059003356 0.0483585411
## [19,] 0.070310579 0.0795313981
## [20,] 0.074333936 0.0846381723
```

```
## [21,] 0.057841055 0.0248706086
## [22,] 0.020153608 0.0105562993
## [23,] 0.000000000 0.0000000000
## [24,] 0.003433300 0.0008401413
## [25,] 0.001671565 0.0079410029
## [26,] 0.002119459 0.0024234978
## [27,] 0.137039841 0.1683565010
```

3. Test des paramètres obtenus par l'algorithme de Baum-welch avec Viterbi, initialisation avec les informations des exercices précédents

Dans cette partie, je teste les paramètres obtenus par **l'algorithme de Baum-Welch**, je fais appel à **l'algorithme de Viterbi** avec comme A et B les paramètres données par **l'algorithme de Baum-Welch**.

```
M<-diag(rep(1,K))-A_baum_welch
M[,K]<-rep(1,K)
Pi_baum_welch<-solve(t(M),b=c(0,1))

viterbi_baum_welch = Viterbi(Pi_baum_welch, A_baum_welch , B_baum_welch,observation)


# je transforme les 1 en -1 et les 2 en 1
viterbi_baum_welch_class_transformed = transformTab(viterbi_baum_welch)


## Affichage du résultat
print("**************************")
```

```
## [1] "**************************"
```

```
cat("\n")
```

```
indicesFrench_ = get_language(viterbi_baum_welch, 1)
indicesEnglish_ = get_language(viterbi_baum_welch, 2)
cat("\n")
```

```
print("**************************Les différents passages en français detectés par viterbi*************
```

```
## [1] "**************************Les différents passages en français detectés par viterbi*************
```

```
cat("\n")
```

```
for (i in 1:length(indicesFrench_)) {
  aAfficher1 = paste("Du caractère ", toString(indicesFrench_[[i]][1]),"à ", toString(indicesFrench_[[
  aAfficher2 = paste(observation[indicesFrench_[[i]][1]:indicesFrench_[[i]][2]], collapse = "")
  print(aAfficher1)
  print(aAfficher2)
  cat("\n")
}
```

```
## [1] "Du caractère  1  à  73"
## [1] "cest ainsi quil restructure la destructuration circonstancielle du nativi"
##
## [1] "Du caractère  160  à  241"
## [1] "cependant mitiger ce raisonnement car il envisage lexpression generative du nativi"
##
```

```
## [1] "Du caractère  298  à  388"
## [1] "ais il ne faut pas oublier pour autant quil envisage la destructuration empirique du nativi"
```

```
cat("\n")
```

```
print("**************************Les différents passages en anglais detectés par viterbi**************
```

```
## [1] "**************************Les différents passages en anglais detectés par viterbi************
```

```
cat("\n")
```

```
for (i in 1:length(indicesEnglish_)) {
  aAfficher1 = paste("Du caractère ", toString(indicesEnglish_[[i]][1]),"  à ", toString(indicesEnglish_
  aAfficher2 = paste(observation[indicesEnglish_[[i]][1]:indicesEnglish_[[i]][2]], collapse = "")
  print(aAfficher1)
  print(aAfficher2)
  cat("\n")
}
```

```
## [1] "Du caractère  74  à  159"
## [1] "sme this is the price and the promise of citizenship  of americas the coldest country "
##
## [1] "Du caractère  242  à  297"
## [1] "sme the year of americas birth in the coldest of month m"
##
## [1] "Du caractère  389  à  443"
## [1] "sme homes have been lost jobs shed businesses shuttered"
```

Commentaires

Je remarque qu'avec une idée de l'initialisation des probabilités du HMM que nous recherchons même approximative, l'algorithme de baum-welch converge et je se rapproche des vraies valeurs de A et de B

4. Baum-Welch avec initialisation aléatoire

Dans cette partie, je teste les paramètres estimés par **l'algorithme de Baum-Welch** sur des paramètres initialisés aléatoirement.
Ce test est fait passant ces paramètres à **l'algorithme de Viterbi**.

```
K = 2
p = dim(X)[2]
# Matrice de transition
tmp1 = sample(1:10, K)
tmp2 = sample(1:10, K)
A_new = matrix(c(0.4,0.6,0.6,0.4),K,K)

A_new[1,] = tmp1
A_new[2,] = tmp2
A_new = A_new/apply(A_new, 1, sum)

print("Matrice de transition créée aléatoirement")
```

```
## [1] "Matrice de transition créée aléatoirement"
```

```r
print(A_new)
```

```
##           [,1]      [,2]
## [1,] 0.2307692 0.7692308
## [2,] 0.2222222 0.7777778
```

```r
cat("\n")
```

```r
print("Somme des deux lignes de A_new")
```

```
## [1] "Somme des deux lignes de A_new"
```

```r
print(apply(A_new,1,sum))
```

```
## [1] 1 1
```

```r
cat("\n")
```

```r
# Calcul de la probabilité initiale
M<-diag(rep(1,K))-A_new
M[,K]<-rep(1,K)
Pi_new<-solve(t(M),b=c(0,1))
print("Probabilité initiale obtenue à partir de A_new")
```

```
## [1] "Probabilité initiale obtenue à partir de A_new"
```

```r
print(Pi_new)
```

```
## [1] 0.2241379 0.7758621
```

```r
cat("\n")
```

```r
# Probabilité d'émission calculée dans le classifieur de Markov à partir de A1 et A2 les matrices de tr

initAleatoireB = function(p, K) {
  B = matrix(NA, p, K)

  B1 = sample(1:60, p)
  B2 = sample(61:120, p)
  B1 = B1
  B2 = B2
  print(B1)
  B[,1] = B1
  B[,2] = B2
  B = t(t(B)/apply(B, 2, sum))
  return (B)
}


B_new = initAleatoireB(p, K)
```

```
##  [1] 52 21 53 43 57 26 33 58  6 23 42 41 44 45 18 59 54 11 40 14  2 38 35 28 16
## [26] 36  8
```

```r
print("Probabilité d'émission créée aléatoirement")
```

```
## [1] "Probabilité d'émission créée aléatoirement"
```

```r
print(B_new)
```

```
##              [,1]       [,2]
##  [1,] 0.057585825 0.04595444
##  [2,] 0.023255814 0.04516889
##  [3,] 0.058693245 0.04124116
##  [4,] 0.047619048 0.03849175
##  [5,] 0.063122924 0.04713276
##  [6,] 0.028792913 0.02670856
##  [7,] 0.036544850 0.03574234
##  [8,] 0.064230343 0.03809898
##  [9,] 0.006644518 0.03260016
## [10,] 0.025470653 0.03967007
## [11,] 0.046511628 0.02749411
## [12,] 0.045404208 0.02788688
## [13,] 0.048726467 0.03417125
## [14,] 0.049833887 0.04399057
## [15,] 0.019933555 0.03652789
## [16,] 0.065337763 0.03220738
## [17,] 0.059800664 0.02945797
## [18,] 0.012181617 0.02513747
## [19,] 0.044296788 0.03888452
## [20,] 0.015503876 0.04320503
## [21,] 0.002214839 0.04673998
## [22,] 0.042081949 0.04477612
## [23,] 0.038759690 0.03456402
## [24,] 0.031007752 0.03731343
## [25,] 0.017718715 0.03142184
## [26,] 0.039867110 0.03927730
## [27,] 0.008859358 0.03613511
```

```
cat("\n")
```

```
print("Somme des deux colones de B")
```

```
## [1] "Somme des deux colones de B"
```

```
print(apply(B_new, 2, sum))
```

```
## [1] 1 1
```

```
lettersWithSpace = c(letters, " ")
hmm2 = list(A = A_new, B = B_new, Pi = Pi_new, Symbols = lettersWithSpace)

BW2 = baumWelch(hmm2, observation, 1)

print("Matrice A estimée: ")
```

```
## [1] "Matrice A estimée: "
```

```
print(BW2$hmm$transProbs)
```

```
##           [,1]      [,2]
## [1,] 0.1788289 0.8211711
## [2,] 0.1815699 0.8184301
```

```
cat("\n")
```

```
print("Probabilité d'émission estimée")
```

```
## [1] "Probabilité d'émission estimée"
```

```
print(BW2$hmm$emissionProbs)
```

```
##              [,1]        [,2]
##  [1,] 0.089072789 0.054721561
##  [2,] 0.007994892 0.012015450
##  [3,] 0.061253773 0.033309530
##  [4,] 0.035934083 0.022372525
##  [5,] 0.176880859 0.101451855
##  [6,] 0.014664744 0.010538813
##  [7,] 0.011355756 0.008514302
##  [8,] 0.052770249 0.024159346
##  [9,] 0.026961500 0.101557409
## [10,] 0.001928623 0.002330110
## [11,] 0.000000000 0.000000000
## [12,] 0.051529317 0.024434075
## [13,] 0.043567851 0.023439575
## [14,] 0.079586544 0.054064630
## [15,] 0.032152425 0.045266444
## [16,] 0.036623219 0.013948695
## [17,] 0.013651419 0.005248979
## [18,] 0.036487582 0.058092122
## [19,] 0.101312778 0.068554277
## [20,] 0.040721146 0.087482815
## [21,] 0.003181294 0.051680354
## [22,] 0.018478809 0.015208594
## [23,] 0.000000000 0.000000000
## [24,] 0.002430057 0.002219098
## [25,] 0.003462396 0.004747636
## [26,] 0.002811483 0.002134654
## [27,] 0.055186412 0.172507151
```

```
M<-diag(rep(1,K))-BW2$hmm$transProbs
M[,K]<-rep(1,K)
Pi_<-solve(t(M),b=c(0,1))

viterbi_baum_welch2 = Viterbi(Pi_, BW2$hmm$transProbs , BW2$hmm$emissionProbs,observation)
# print("La sortie de l'algorithme de Viterbi")
# print(viterbi_baum_welch2)
# cat("\n")

# je transforme les 1 en -1 et les 2 en 1
print("on transforme les classes prédites, les 1 en -1 et les 2 en 1")
```

```
## [1] "on transforme les classes prédites, les 1 en -1 et les 2 en 1"
```

```
viterbi_baum_welch_class_transformed2 = transformTab(viterbi_baum_welch2)
print(viterbi_baum_welch_class_transformed2)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [297] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [334] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [371] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [408] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
cat("\n")
```

Avec une initialisation aléatoire, j'ai pas les bons paramètres correspondant à notre séquences

5. Commentaires

Je vois bien que l'algorithme de baum-welch présente cependant le défaut de ne pas toujours converger vers la même solution, voire même de ne pas converger du tout sous certaines conditions.

Et ces problèmes surviennent surtout losque nous n'avons aucune idée des probabilités du HMM que nous recherchons (pas même approximative) et que nous commençons donc avec un HMM avec des paramètres aléatoires (ce qui signifie que les probabilités de départ et de transition sont aléatoires).

Exercice 6: Modèle à bloc stochastique

1. Distance euclidienne d(i, j) entre tous les textes deux à deux

```
# Fonction distance euclidienne
distEucl = function(val1, val2) {
  n = length(val1)
  somme = 0
  for (i in 1:n) {
    somme = somme + (val1[i] - val2[i])^2
  }
  distance = sqrt(somme)
  return (distance)
}

# Fonction création d'une matrice de valeur nulle sur la diagonale, représentatnt la distance euclidien

createMatriceDist = function(mat) {
  n = dim(mat)[1]
  newMat = matrix(0,n,n)
  for (i in 1:n) {
    for (j in 1:n) {
      newMat[i,j] = distEucl(mat[i,], mat[j,])
    }
  }
  return (newMat)
}
matriceDist = createMatriceDist(X)
# print(matriceDist)
```

2. Création d'une matrice binaire K

```
# La médiane des distances calculées
medianMat = median(matriceDist)

print("La médiane des distances calculées")
```

```
## [1] "La médiane des distances calculées"
```

```
print(medianMat)
```

```
## [1] 0.07265913
```

```
# Matrice K

n = dim(X)[1]
K = matrix(0, n, n)
K = matrix(as.integer(matriceDist < medianMat), n, n)

# print("La matrice binaire K")
# print(K)
```

##j'affiche un graphique d'adjacence entre nos textes à partir de K

Dans cette partie, j'affiche un graphique à partir de K. Vous pourrez vérifier que ça marche,j'affiche à côté les indices des textes par langues dans notre base de texte

```
# Fonction qui permet de récupérer les indices des textes par langue

getIndicesByLangues = function(intialTexts, y, langClass) {
  indices = which(y == langClass)
  # textByLangues = intialTexts[indices]
  return (indices)
}

tmpY = y

indice1 = which(y == -1)
indice2 = which(y == 1)
tmpY[indice1] = 0

indicesFrançais = getIndicesByLangues(intialTexts, y, -1)
indicesAnglais = getIndicesByLangues(intialTexts, y, 1)
cat("\n")
```

```
cat("\n")
```

```
print("Français")
```

```
## [1] "Français"
```

```
print(indicesFrançais)
```

```
##  [1]  1  2  3  7 10 12 13 14 15 16 18 20 21 23 24 33 34 36 38 39 46 49 50 52 53
## [26] 54 56 61 62 64 65 68 70 73 75 77 79 80 81 82 83 87 88 89 90
```

```
print("Anglais")
```

```
## [1] "Anglais"
```

```
print(indicesAnglais)
```

```
##  [1]  4  5  6  8  9 11 17 19 22 25 26 27 28 29 30 31 32 35 37 40 41 42 43 44 45
## [26] 47 48 51 55 57 58 59 60 63 66 67 69 71 72 74 76 78 84 85 86
```

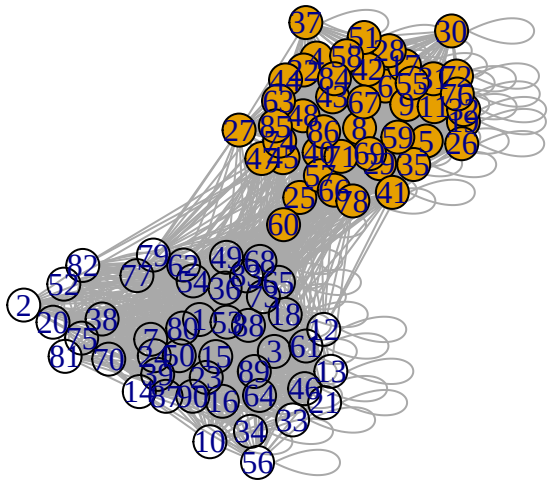
```
cat("\n")
```

```
print("Les textes en anglais sont colorés et les textes en français sont sans couleur")
```

```
## [1] "Les textes en anglais sont colorés et les textes en français sont sans couleur"
```

```
plot(graph_from_adjacency_matrix(K,mode="undirected"),vertex.color=tmpY, main="Textes en anglais colorés
```

Textes en anglais colorés en orange et ceux en français sont sans cou



3. Modèles à blocs stochastiques

```
class.ind<-function (cl)
{
  n <- length(cl)
  cl <- as.factor(cl)
  x <- matrix(0, n, length(levels(cl)))
  x[(1:n) + n * (unclass(cl) - 1)] <- 1
  dimnames(x) <- list(names(cl), levels(cl))
  x
}


VEMaffiliation<-function(mat,Q=2,equal.proportions=TRUE,nbstart=10,max.iter=100,epsilon=1e-6){
  parameters<-list(alpha=0.1,beta=0.1,pi=rep(1/Q,Q))
  bestJ <- -Inf
  for (run in 1:nbstart){
    # Initialisation of the partition
    initialisation<-kmeans(mat,Q,nstart=5)
    # print("initalisation")
    # print(initialisation$cluster)
```

```
    Tau<-class.ind(initialisation$cluster)
    J <- -Inf
    for (it in 1:max.iter){
      J.old<-J
      resM<- Mstep(mat,Tau,parameters,equal.proportions=equal.proportions)
      parameters<-resM$parameters
      resE<- VEstep(mat,Tau,parameters)
      Tau<-resE$Tau
      J<-resE$J
      if (abs(J-J.old)< epsilon) break
    }
    if (J > bestJ) {
      bestJ<-J
      bestParameters <- resM$parameters}
  }
  return(list(Tau=resE$Tau,parameters=bestParameters,J=bestJ))
}

Mstep <- function(mat, Tau,equal.proportions=TRUE, parameters=NULL) {
  ## INTIALIZE
  eps<-1e-6
  Q <- ncol(Tau) # num classes
  n <- nrow(Tau) # num nodes
  nql <- matrix(0,Q,Q)
  theta.ijql <- matrix(0,n,n)
  u.trig <- upper.tri(mat)
  v.trig <- upper.tri(nql)
  nodiag <- upper.tri(mat) | lower.tri(mat)
  for (q in 1:Q) {
    for (l in q:Q) {
      ## intermediary calculation
      theta.ijql <- Tau[,q] %*% t(Tau[,l])
      diag( theta.ijql)<-0

      ## nql is the number of edges between class q and class l
      nql[q,l] <- sum(theta.ijql*mat)
    } # next l
  } # next q
  nq<-colSums(Tau)
  if (equal.proportions==TRUE)
    pi<- rep(1/Q,Q)
  else {
    pi<-nq/n
  }
  nqnl<- cbind(nq) %*% rbind(nq)
  beta<- max(sum(nql[v.trig]) / sum(nqnl[v.trig]),eps)
  alpha <- max(sum(diag(nql))/sum(nq*(nq-1)),eps)
  J <- JRx(Tau, mat, parameters)
  return(list(parameters=list(pi=pi,alpha=alpha,beta=beta),J=J))
}

VEstep <- function( mat,
                    Tau,
```

```
                    parameters) {
# Variationnal E step of the EM algorithm for Bernoulli MixNet Model
# INPUT
# mat : adjacency matrix
# Tau : Matrix of conditionnal probabilities
#
# OUTPUT
# Tau
FP.maxIt = 50
eps = 1e-6
verbose = FALSE
if (verbose==TRUE) {
  cat("\n - Fixed point resolution... ")
}
## ====================================
## INITIALIZING
n <- nrow(Tau) # num nodes
Q <- ncol(Tau) # num classes
pi<-parameters$pi
logpi <- pmax(log(pi),log(.Machine$double.xmin))
beta <- parameters$beta
alpha <- parameters$alpha
alpha.ql<-matrix(beta,Q,Q)
diag(alpha.ql)<- alpha
ones <- cbind(rep(1, n))
class.names <- colnames(Tau)
## ====================================
## SOLVING TAU WITH FImatED POINT ALGO
## convergence setup
J <- -Inf # JRx criterium
E.Delta <- Inf # convergence deltas
E.Deltas <- c()
E.It <- 0 # iterations
convergence <- list( JRx.pb=FALSE, Iteration.pb=FALSE, Converged=FALSE, Pb=FALSE)
## convergence loop
if (verbose) { cat(" iterate: ") }
while ( (convergence$Pb==FALSE) && (convergence$Converged==FALSE) ) {
  E.It <- E.It+1
  Tau.old <- Tau
  J.old <- J
  if (verbose) { cat(" ",E.It) }
  ## prep current estimate of log(Tau)
  logTau <- matrix(0,n,Q)
  for (q in 1:Q) {
    for (l in 1:Q) {
      ## normal Lagrangians
      Beta.ijql <- mat * log(alpha.ql[q,l]) + (1-mat) * log(1-alpha.ql[q,l])
      diag(Beta.ijql) <- 0
      logTau[,q] <- logTau[,q] + apply(((ones %*% Tau[,l]) * Beta.ijql), 1, sum )
    } # next l
  } # next q
  ## Normalizing in the log space to avoid numerical problems
  logTau <- logTau - apply(logTau,1,max)
```

```
    ## Now going back to exponential with the same normalization
    Tau <- (matrix(1,n,1) %*% pi) * exp(logTau)
    Tau <- pmin(Tau,.Machine$double.xmax)
    Tau <- pmax(Tau,.Machine$double.xmin)
    Tau <- Tau / (rowSums(Tau) %*% matrix(1,1,Q))
    Tau <- pmin(Tau,1-.Machine$double.xmin)
    Tau <- pmax(Tau,.Machine$double.xmin)

    J <- JRx(Tau, mat, parameters)
    # convergence pb : JRx decreases
    convergence$JRx.pb <- (J < J.old)
    if (convergence$JRx.pb==TRUE) {
      Tau <- Tau.old
    }
    ## Delta criterium
    E.Delta <- J - J.old
    E.Deltas[E.It] <- E.Delta
    ## convergence ?
    convergence$Iteration.pb <- (E.It > FP.maxIt) # have we hit iter.max ?
    convergence$Converged <- (abs(E.Delta) < eps ) # has the algo converged ?
    convergence$Pb <- ( convergence$Iteration.pb || convergence$JRx.pb)
  } # repeat till convergence or itMax
  ## probabilistic class estimation
  colnames(Tau) <- class.names
  ## check non-convergence
  if ( convergence$Pb==TRUE ) {
    if (verbose) {
      cat(" can't enhance the criteria anymore...\n" )
    }
  }
  return(list(Tau=Tau,J=J))
}

JRx <- function (Tau, mat, parameters) {
  beta <- parameters$beta
  alpha <- parameters$alpha
  pi <-parameters$pi
  Q<-length(pi)
  alpha.ql <- matrix(beta, Q,Q)
  diag(alpha.ql)<- alpha
  if (is.null(parameters)) {
    cat("\n JRx : WARNING : null parameters argument, returning NULL !!!\n")
    return(NULL)
  }
  n <- dim(Tau)[1]
  Q <- dim(Tau)[2]

  u.tri <- upper.tri(mat)
  ## Doit augmenter au cours des it??rations
  logTau <- pmax(log(Tau),log(.Machine$double.xmin))
  logpi <- pmax(log(pi),log(.Machine$double.xmin))
  ## Les 2 premiers termes de J...
  J <- - sum( Tau * logTau ) + sum ( Tau %*% logpi )
```

```
  eps <- .Machine$double.xmin
  ## ... et le 3??me
  for (q in 1:Q) {
    for (l in 1:Q) {
      lnf.ijql <- mat * log(alpha.ql[q,l]) + (1-mat) * log(1-alpha.ql[q,l])
      tau.ijql <- Tau[,q] %*% t(Tau[,l])
      J <- J + sum( tau.ijql[u.tri] * lnf.ijql[u.tri] )
    } # next l
  } # next q
  return(J)
}
```

Vérification performance du MBS

```
res.VEM<-VEMaffiliation(K,Q=2,equal.proportions=TRUE,nbstart=10,max.iter=100,epsilon=1e-6)

print(res.VEM$parameters)
```

```
## $pi
## [1] 0.5 0.5
##
## $alpha
## [1] 0.8888889
##
## $beta
## [1] 0.108642
```

```
predict_class_mbs<-apply(res.VEM$Tau,1,which.max)

matConfusion_mbs = ConfusionMatrix(y, predict_class_mbs)
print("Matrice de confusion du MBS")
```

```
## [1] "Matrice de confusion du MBS"
```

```
print(matConfusion_mbs)
```

```
##     y_pred
## y     1  2
##   -1  0 45
##   1  45  0
```

```
print("vrai y")
```

```
## [1] "vrai y"
```

```
print(y)
```

```
##  [1] -1 -1 -1  1  1  1 -1  1  1 -1  1 -1 -1 -1 -1 -1  1 -1  1 -1 -1  1 -1 -1  1
## [26]  1  1  1  1  1  1  1 -1 -1  1 -1  1 -1 -1  1  1  1  1  1  1 -1  1  1 -1 -1
## [51]  1 -1 -1 -1  1 -1  1  1  1  1 -1 -1  1 -1 -1  1  1 -1  1 -1  1  1 -1  1 -1
## [76]  1 -1  1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1 -1
```

```
print("y prédite")
```

```
## [1] "y prédite"
```

```
print(predict_class_mbs)
```

```
##  [1] 2 2 2 1 1 1 2 1 1 2 1 2 2 2 2 2 1 2 1 2 2 1 2 2 1 1 1 1 1 1 1 2 2 1 2 1 2
## [39] 2 1 1 1 1 1 1 2 1 1 2 2 1 2 2 2 1 2 1 1 1 1 2 2 1 2 2 1 1 2 1 2 1 1 2 1 2 1
## [77] 2 1 2 2 2 2 2 1 1 1 2 2 2 2
```

L'algorithme arrive à séparer les textes des deux langues

J'ai parfois une interversion des paramètres, probablement dû à l'initialisation des paramètres du VEM.

CONCLUSION

Différences parmi les classifieurs étudiés :

- Le classifieur de bayes basés sur des unigrammes
- Le classifieur markovien (pas les mêmes resultats à chaque fois dû au caractères aléatoires des simulations des chaînes de Markov)
- Viterbi donne une bonne performance sur les séquences observées avec les bons paramètres (ici nos probabilités d'emission des caractères et de la matrice de transition)
- Baum-Welch marche mieux quand j'initialise avec les bons paramètres (ici nos 2 matrices de transition CF exercice3), tandis que si je l'initialise aleatoirement, j'ai parfois peu ou pas de convergence vers une solution correspondante à nos observations
- VEM basé sur la matrice de similarité, si des mots ont des frequences qui se rapprochent mais sont de deux langues differentes, je peux avoir le risque d'avoir une mauvaise classification

Avec une base de textes bien fournis et des textes mieux structurés, je peux affirmer que le classifieur de bayes et le VEM sont les plus performants.