Projet - Reconnaissance de langues écrites

Unsupervised Learning

M2 Data Science

Etienne Gaucher

1 - Base de textes

1.1 Base de fichiers & dataset

Nous avons constitué une base de fichiers de 45 textes, dont 18 sont en anglais et 27 sont en français. La base de fichiers se situe dans le dossier `texts` . Dans la suite de l'exercice, on cherche à construire un tableau

de fréquence de caractère pour chaque langue. Un traitement des textes s'impose auparavant, car nous disposons uniquement des textes bruts. On réalise les étapes suivantes pour chaque texte afin d'obtenir des textes "nettoyés" :

- on élimine les signes de ponctuation
- on supprime les caractères spéciaux comme les backslash, antislash, apostrophe, le symbole degré...
- on supprime les éventuels chiffres
- on remplace les lettres accentuées par les lettres non accentuées et le c cédille par le c basique
- on remplace les majuscules par des minuscules

Nous avons choisi de remplacer les lettres accentuées car sinon, la reconnaissance de langues est élémentaire étant donné qu'il n'y a pas d'accent dans la langue anglaise. La problématique serait facilement résolue, notre choix apporte donc plus de défi.

On importe les textes sous forme d'un dataframe $(x_i, y_i)_{i=1,\dots,n}$ où $\forall i$, x_i est le texte et y_i la langue du texte. Les textes en anglais correspondent à $y_i = 1$, tandis que les textes en français sont caractérisés par $y_i = -1$.

```
# Import de tidyverse pour utiliser le pipe %>%
library(tidyverse)

# Import de tm pour utiliser des fonctions sur les textes
library(tm)

recup_data <- function(path = "./texts/")
{
  # Récupère les fichiers du path pour construire le dataframe (X,Y) après traitement des textes
```

```r
# Input : chemin relatif (depuis le dossier où est stocké le .rmd) vers le dossier contenant les t

# Output : dataframe décrit précédemment

# Définition de y (les 18 premiers textes du dossier sont en anglais, les 27 autres sont en franç
vector_lang <- c(rep(1,18),rep(-1,27))

# Liste des textes
L_text <- list()

i <- 1

# Boucle for sur l'ensemble des fichiers .txt présents dans le dossier path
for(t in list.files(path, pattern="txt")){
  L_text[[i]] <- readLines(paste("./texts/",t, sep = ""), encoding = 'UTF-8') %>%
    paste(collapse=" ") %>%
    unlist %>%

    # Supprime les caractères spéciaux et les signes de ponctuation
    str_remove_all("[,?:;°.-…!–‘“/”'’\"-]") %>%
    str_remove_all("\\(") %>%
    str_remove_all("\\)") %>%
    str_remove_all("\\«") %>%
    str_remove_all("\\»") %>%
    str_remove_all("\\[") %>%
    str_remove_all("\\]") %>%

    # Retire les chiffres
    removeNumbers() %>%
```

```
      # Transforme les majuscules en minuscules
      str_to_lower() %>%

      # Remplace les lettres accentuées
      chartr(old = "àáâäãèéêëìíîïòóôöõùúûüýÿçñœæ", new = 'aaaaaeeeeiiiioooooouuuuyycnea')

      i <- i+1
  }

  return(data.frame(
    x = unlist(L_text),
    y = vector_lang
    ))
}

# Création du dataset avec les textes
dataset <- recup_data()

# Dimension
dim(dataset)
```

```
## [1] 45  2
```

1.2 Tableau des log-fréquences X

Une fois que chaque texte a subi le traitement et que le dataset est créé, on peut passer à l'étape suivante : les fréquences. On veut connaître la fréquence de chaque lettre pour chacun des textes. Cependant, on ne va pas calculer exactement la fréquence de chaque lettre, mais $\log(1+f)$ avec f la fréquence. Le symbole

espace n'est pas pris en compte ici. La fonction `table()` renvoie le nombre d'occurrences de chaque lettre pour un texte donné. Mais certaines lettres ne sont pas présentes dans tous les textes, d'où la fonction `normalize1` qui ajoute des occurrences nulles pour les lettres absentes.

La fonction `table_of_dataset` retourne le tableau des occurrences de chaque lettre pour tous les textes, ce qui permet ensuite de trouver les log-fréquences à l'aide de la fonction `matrix_logfrequency` .

```
normalize1 <- function(table_letter, names_character){
  # Ajout des caractères manquants à la table d'effectif des caractères d'un texte (occurrences null

  # Inputs :
  # "table_letter" : Tableau d'effectif des caractères d'un texte
  # "names_character" : Ensemble des caractères possibles

  # Output :
  # "res" : Tableau complet d'effectif de chaque caractère (dans un texte)

  N <- length(names_character)

  # Si aucun caractère ne manque, on retourne le tableau
  if (length(table_letter)==N){ return(table_letter) }

  else{
    # Création d'un vecteur nul
    res <- rep(0,N)

    names(res) <- names_character

    # Remplace les occurrences nulles par les valeurs de table_letter lorsque le caractère est prése
```

```r
    for (character in names(table_letter))
    {
      res[character] <- table_letter[character]
    }

    return (res)
  }
}

table_of_dataset <- function(dataset){
  # Tableau d'effectif par texte

  # Input :
  # "dataset" : Dataset [Dataframe (X,Y)]

  # Output :
  # "table.res" : Tableau d'effectif de chaque caractère pour chaque texte + dernière colonne indiqu

  # Vecteur des lettres
  character_list <- c("a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p",

  # Nombre de textes dans la liste
  n <- length(dataset$x)
  # Nombre de lettres
  m <- length(character_list)

  table.res <- matrix(0,n,m+1)
  colnames(table.res) <- c(character_list, "lang")

  # Boucle for sur les textes de la liste
```

```r
  for (t in (1:n))
  {
    # Création d'une table avec le nombre d'occurrences de chaque lettre du texte
    table_letter <- strsplit(as.vector(dataset$x[t]), split="") %>% unlist() %>% table()

    # On enlève le nombre d'espace et on ajoute les éventuelles lettres manquantes avec la fonction n
    table_letter <- normalize1(table_letter[-1], character_list)

    # On ajoute la dernière colonne "lang"
    table.res[t,] <- c(table_letter, dataset$y[t])
  }

  return(table.res)
}

# Test
#table = table_of_dataset(dataset)
#print(table)

matrix_logfrequency <- function(dataset){
  # Tableau des log-fréquences des caractères pour chaque texte

  # Input :
  # "dataset" : Dataset [Dataframe (X,Y)]

  # Output :
  # "matrix.res" : Tableau de la log-fréquence de chaque caractère pour chaque texte + dernière colo

  X <- table_of_dataset(dataset)
  n <- dim(X)[2]
```

```r
  # Calcul des log-fréquences de chaque texte
  matrix.res <- cbind(log(1 + (X[,-n] / rowSums(X[,-n]))), X[,n])

  return(matrix.res)
}

# Resultat Q2
logf <- matrix_logfrequency(dataset)
head(logf)
```

```
##               a          b          c          d         e          f
## [1,] 0.07985998 0.01415118 0.03117945 0.03882475 0.1185745 0.02192727
## [2,] 0.08551235 0.01427746 0.03416115 0.02893618 0.1217032 0.02075378
## [3,] 0.08102324 0.01489231 0.02810836 0.03609968 0.1256705 0.01747581
## [4,] 0.08691394 0.01354772 0.03107866 0.03973021 0.1056722 0.03070080
## [5,] 0.07904562 0.01287250 0.03406223 0.04276046 0.1154033 0.02116639
## [6,] 0.07257363 0.01285440 0.02439854 0.03439148 0.1159191 0.01979686
##               g          h          i            j           k          l
## [1,] 0.01649036 0.06809513 0.06439005 0.0000000000 0.007100622 0.02964336
## [2,] 0.01840361 0.03300241 0.06495036 0.0017958700 0.001795870 0.04967425
## [3,] 0.02005266 0.05296107 0.07686379 0.0011246487 0.012672551 0.03898986
## [4,] 0.01968036 0.04570483 0.06888894 0.0007791196 0.006216026 0.03070080
## [5,] 0.02205095 0.04131600 0.07904562 0.0012044566 0.006307274 0.03347965
## [6,] 0.02697771 0.04597190 0.07093160 0.0008817047 0.005571045 0.05101393
##               m          n          o          p            q          r
## [1,] 0.02037688 0.06439005 0.06513217 0.01726887 0.0015822788 0.07985998
## [2,] 0.03126176 0.08220656 0.07667251 0.02485335 0.0000000000 0.06382676
## [3,] 0.02956609 0.07721107 0.05722093 0.01784434 0.0011246487 0.05863686
```

```
## [4,] 0.02463558 0.07722082 0.06487946 0.02387483 0.0015576327 0.05607530
## [5,] 0.02352349 0.08210330 0.06728407 0.01673179 0.0009034785 0.05936548
## [6,] 0.02094927 0.07147924 0.07612216 0.02726387 0.0011754336 0.05991524
##               s          t          u           v          w           x
## [1,] 0.05768612 0.08423635 0.02964336 0.012588679 0.03117945 0.000000000
## [2,] 0.06213900 0.08441164 0.02718844 0.012503884 0.01368662 0.002393777
## [3,] 0.06673977 0.08033120 0.02628321 0.009705190 0.01710715 0.004117543
## [4,] 0.07794206 0.09580605 0.02767279 0.010081512 0.01354772 0.002724267
## [5,] 0.07318214 0.08432121 0.03493547 0.006307274 0.01554590 0.002407464
## [6,] 0.07093160 0.08453355 0.02611872 0.013434782 0.01979686 0.001469076
##               y            z
## [1,] 0.01337023 0.0007914524 1
## [2,] 0.01131975 0.0000000000 1
## [3,] 0.01636941 0.0000000000 1
## [4,] 0.01239365 0.0003896357 1
## [5,] 0.01227744 0.0003012502 1
## [6,] 0.01430473 0.0000000000 1
```

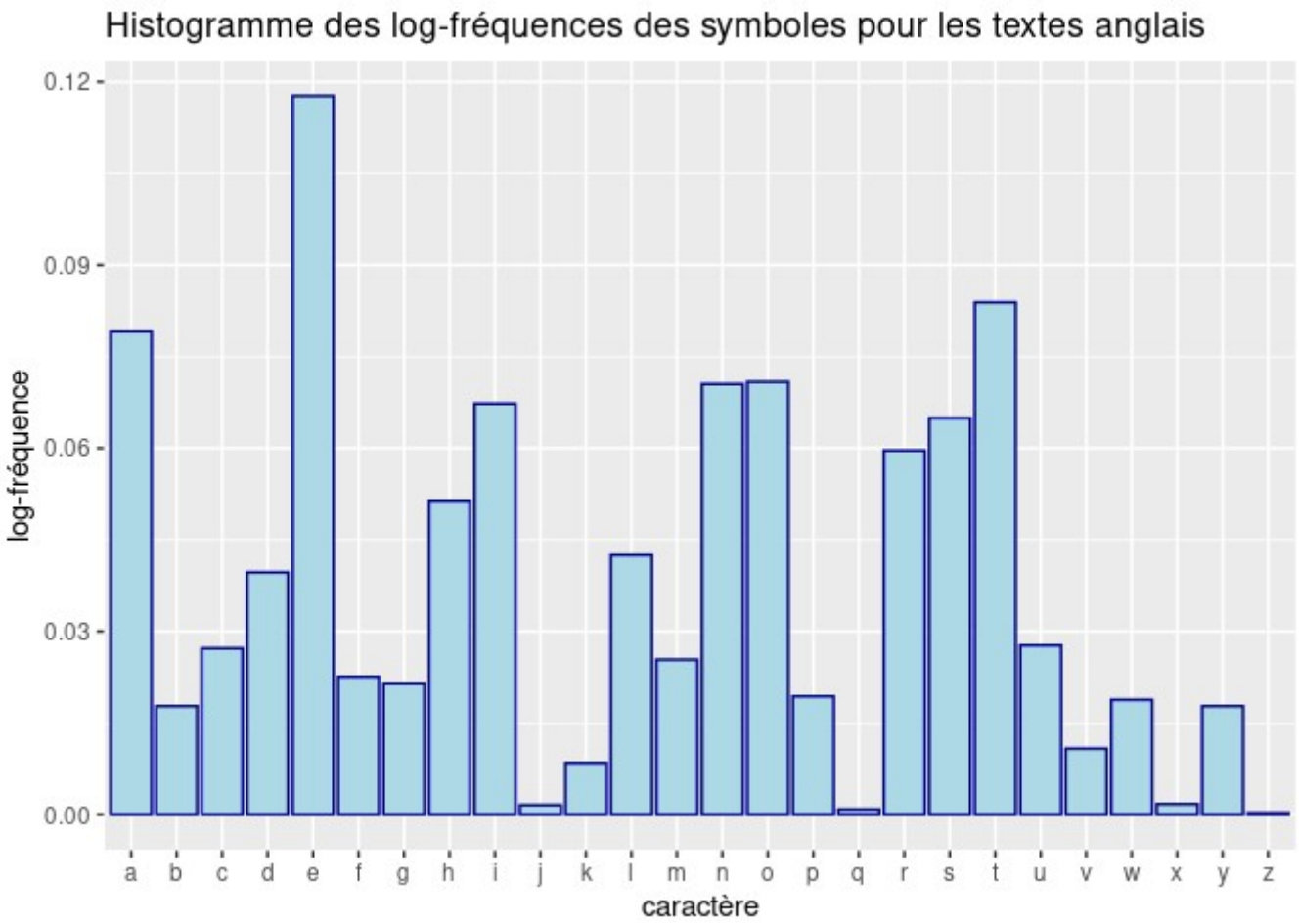
1.3 Histogrammes des log-fréquences des symboles

On veut tracer l'histogramme des log-fréquences des caractères pour les textes en anglais et les textes en français. On ne peut pas faire la moyenne des log-fréquences obtenues à la question 2, car cela est mathématiquement incorrect. On doit redémarrer des effectifs de chaque texte, et les sommer selon la langue avant de calculer la log-fréquence.

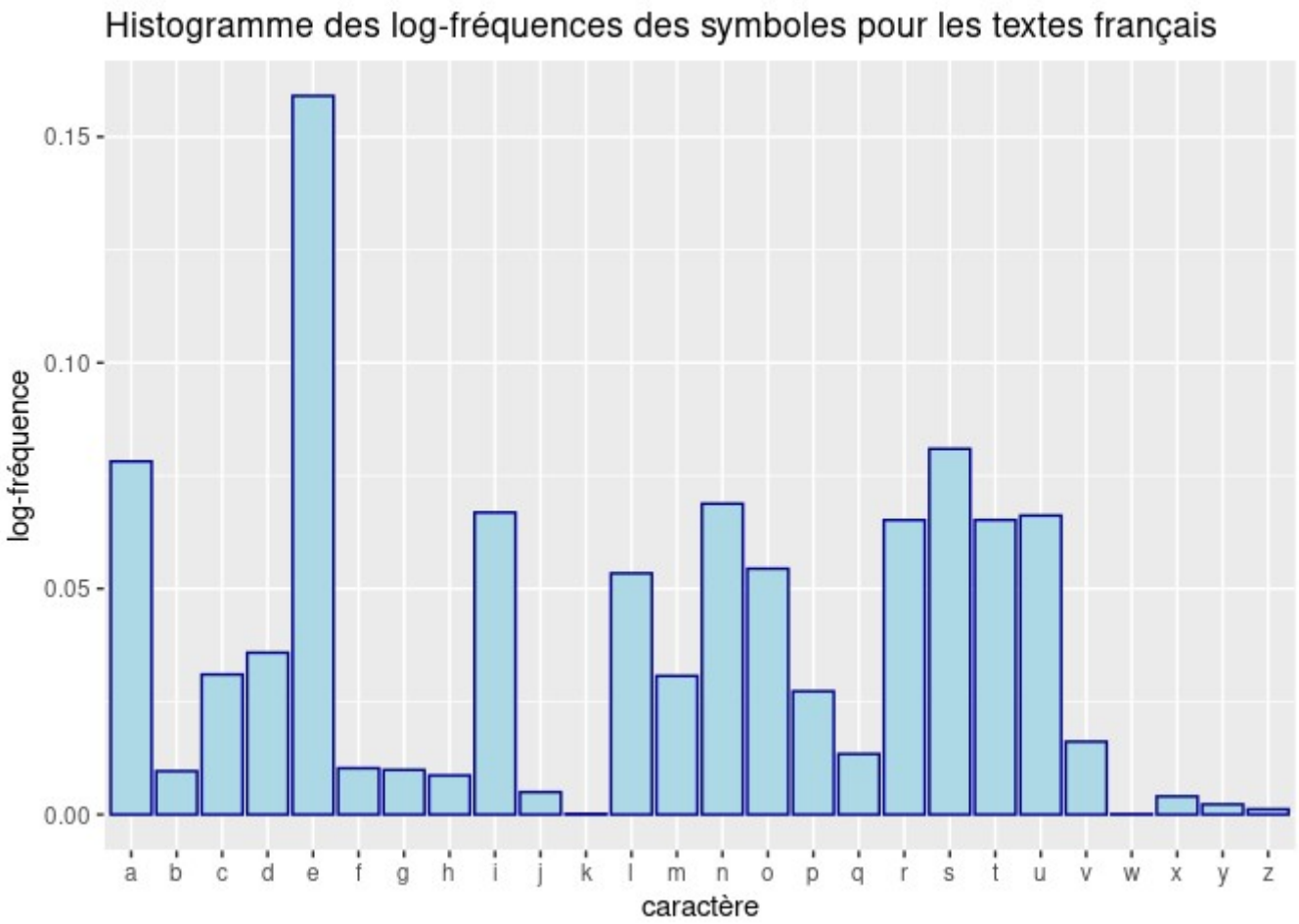
```
# Import de ggplot2 pour tracer les histogrammes
library(ggplot2)
```

```r
prepare_for_hist <- function(dataset){
  # Préparation des données pour l'histogramme

  # Input :
  # "dataset" : Dataset [Dataframe (X,Y)]

  # Output :
  # "dataf_hist" : Dataframe à 3 colonnes (caractère - fréquence du caractère - langue associée à la

  X <- table_of_dataset(dataset)
  n <- dim(X)[2]

  # Tableau des log-fréquences des caractères pour chaque langue
  X_fr <- as.vector(colSums(X[which(X[,n]==-1),-n]))
  X_fr <- log(1 + (X_fr / sum(X_fr)))
  X_en <- as.vector(colSums(X[which(X[,n]==1),-n]))
  X_en <- log(1 + (X_en / sum(X_en)))

  # Dataframe à 3 colonnes (caractère - fréquence du caractère - langue associée à la fréquence) et
  caract <- rep(colnames(X)[-n],2)
  freq <- c(X_fr,X_en)
  lang <- c(rep(-1,n-1),rep(1,n-1))
  dataf_hist <- data.frame("caractere" = caract,
                           "frequence" = freq,
                           "langage" = lang)

  return(dataf_hist)
}

# Résultat Q3 :
```

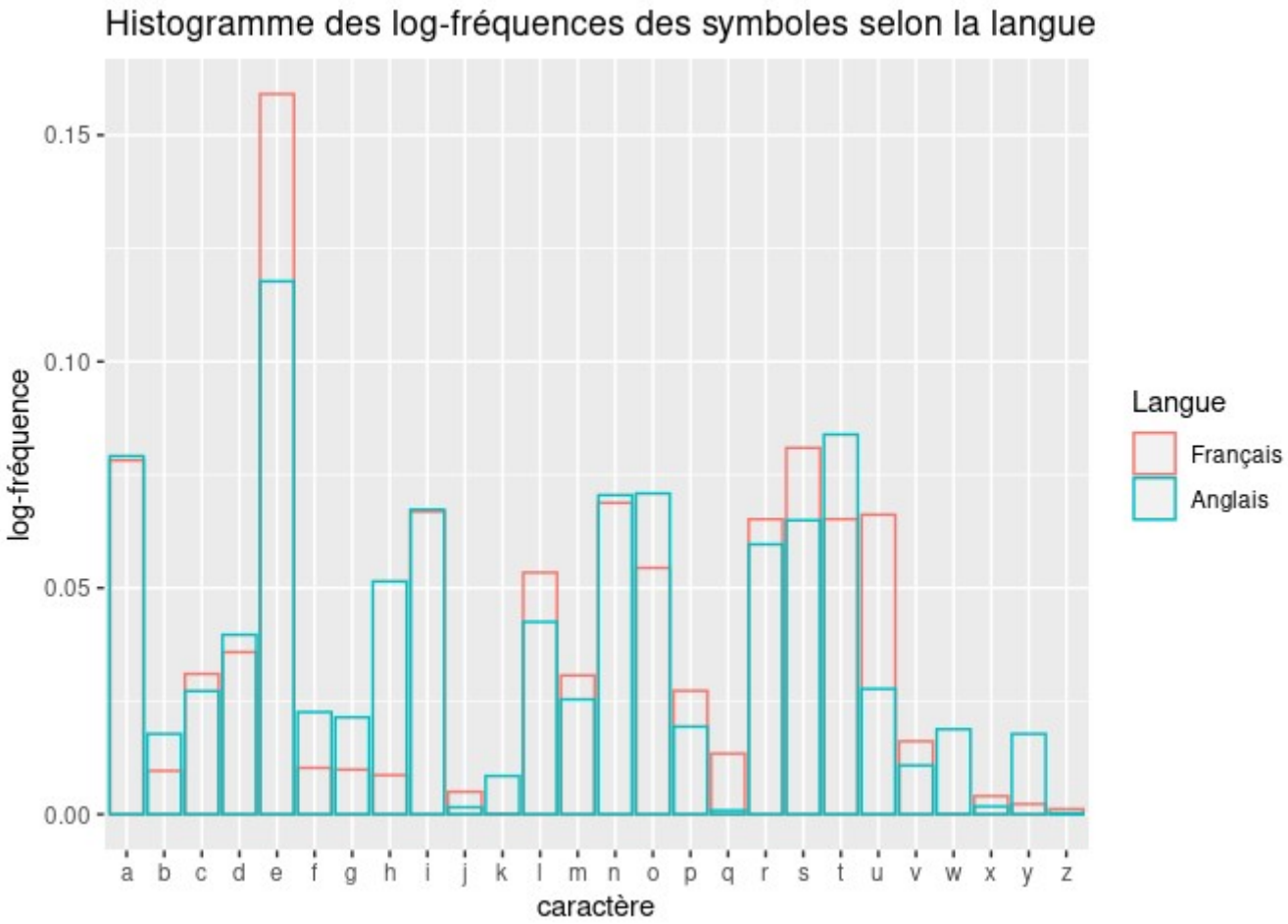
```r
dataf_hist <- prepare_for_hist(dataset)
# Test : print(dataf_hist)


# Histogramme pour les textes anglais
ggplot(dataf_hist[dataf_hist$langage == 1,], aes(x=caractere, y=frequence)) +
  geom_bar(stat="identity", color="darkblue", fill="lightblue") +
  labs(title="Histogramme des log-fréquences des symboles pour les textes anglais") +
  xlab("caractère") +
  ylab("log-fréquence")
```




```r
# Histogramme pour les textes français
ggplot(dataf_hist[dataf_hist$langage == -1,], aes(x=caractere, y=frequence)) +
  geom_bar(stat="identity", color="darkblue", fill="lightblue") +
  labs(title="Histogramme des log-fréquences des symboles pour les textes français") +
  xlab("caractère") +
  ylab("log-fréquence")
```




```r
# Histogramme de comparaison
ggplot(dataf_hist, aes(x=caractere, y=frequence, color = as.factor(langage))) +
  geom_bar(stat="identity", position=position_identity(), fill ="NA") +
  labs(title="Histogramme des log-fréquences des symboles selon la langue") +
  xlab("caractère") +
  ylab("log-fréquence") +
  scale_color_discrete(name = "Langue", labels = c("Français", "Anglais"))
```



Le dernier histogramme nous montre que des lettres sont beaucoup plus fréquentes dans une langue que l'autre. Par exemple, les lettres e, q, s et u sont plus courantes en français, alors que les lettres h, w, et y sont plus courantes en anglais. Notre classifieur de Bayes naïf va utiliser ces propriétés pour prédire la langue d'un texte.

2 - Classifieur de Bayes naïf

2.1 Estimation des paramètres

On cherche à construire un classifieur de Bayes naïf pour prédire la langue d'un texte. Notre première tâche consiste à estimer les paramètres du modèle, c'est-à-dire les moyennes et les variances pour chaque classe du jeu d'apprentissage. Dans notre cas, il y a deux classes : anglais et français.

Les notations utilisées sont les suivantes :

- $\hat{p}_{anglais}$ probabilité marginale que le texte soit en anglais. On l'estime par $\hat{p}_{anglais} = \frac{1}{n} \sum_{i=1}^{n} 1_{y_i=1}$ avec n le nombre de textes dans le jeu d'apprentissage, et y la langue du texte
- $\hat{p}_{français}$ probabilité marginale que le texte soit en français. On l'estime par $\hat{p}_{français} = \frac{1}{n} \sum_{i=1}^{n} 1_{y_i=-1}$ avec n le nombre de textes dans le jeu d'apprentissage, et y la langue du texte
- $x = \begin{pmatrix} \log(f_a + 1) \\ \vdots \\ \log(f_z + 1) \end{pmatrix} \in \mathbb{R}^{26}$ vecteur des log-fréquences du texte à classifier
- $\mu_{anglais} = \begin{pmatrix} \mu_{\text{log\_fréq a, anglais}} & \cdots & \mu_{\text{log\_fréq z, anglais}} \end{pmatrix} \in \mathbb{R}^{26}$ moyennes des log-fréquences des textes en anglais du jeu d'apprentissage
- $\mu_{français} = \begin{pmatrix} \mu_{\text{log\_fréq a, français}} & \cdots & \mu_{\text{log\_fréq z, français}} \end{pmatrix} \in \mathbb{R}^{26}$ moyennes des log-fréquences des textes en français du jeu d'apprentissage
- $\sigma^2_{anglais} I_p \in \mathbb{R}^{26 \times 26}$ variances des log-fréquences des textes en anglais du jeu d'apprentissage

- $\sigma^2_{français}\, I_p \in \mathbb{R}^{26\times 26}$ variances des log-fréquences des textes en français du jeu d'apprentissage
- $\hat{\pi}(anglais|x)$ probabilité que le texte à classifier soit en anglais
- $\hat{\pi}(français|x)$ probabilité que le texte à classifier soit en français

Dans le cadre du modèle de Bayes naïf, on a

$$\hat{\pi}(k|x) = \frac{\hat{p}_k.\, f_k(x|\hat{\theta}_k)}{\sum_{l=1}^{2} \hat{p}_l.\, f_l(x|\hat{\theta}_l)} \propto \hat{p}_k.\, f_k(x|\hat{\theta}_k)$$

On ne calculera donc pas le dénominateur pour prédire la langue d'un texte.

La fonction `compute_param` estime les paramètres de moyennes et variances des classes d'un jeu d'apprentissage. Le vecteur passé en argument correspond aux index des textes du jeu d'apprentissage.

```
compute_param <- function(index_train, matrix_logfreq)
{
  # Liste des paramètres de moyennes et variances des classes d'un jeu d'apprentissage

  # Inputs :
  # "index_train" : vector, index des textes du jeu d'apprentissage
  # "matrix_logfreq" : matrix, matrice des log-fréquences de l'ensemble des textes

  # Output :
  # Moyennes et variances des log-fréquences des textes du jeu d'apprentissage pour les classes angl
```

```
  logf_train <- matrix_logfreq[index_train, ]

  # Estimation des moyennes
  mu_en <- apply(logf_train[logf_train[, ncol(logf_train)] == 1, -ncol(logf_train)], 2, mean)
  mu_fr <- apply(logf_train[logf_train[, ncol(logf_train)] == -1, -ncol(logf_train)], 2, mean)

  # Estimation des variances
  var_en <- apply(logf_train[logf_train[, ncol(logf_train)] == 1, -ncol(logf_train)], 2, var)
  var_fr <- apply(logf_train[logf_train[, ncol(logf_train)] == -1, -ncol(logf_train)], 2, var)

  return(list(
    "mu_en" = mu_en,
    "mu_fr" = mu_fr,
    "var_en" = diag(var_en),
    "var_fr" = diag(var_fr)
  ))
}
```

2.2 Le classifieur

On définit ensuite la fonction `naive_bayes` qui, pour un ensemble de textes passé en argument, renvoie la langue prédite par le classifieur. Puisque les log-fréquences de chaque texte ont déjà été calculées à l'exercice 1, on passera plutôt en argument les index des textes à classifier. Les index des textes constituants le jeu d'apprentissage sont également un argument de la fonction, ce qui facilitera l'implémentation de la validation croisée dans la question suivante.

```
# Import de mvtnorm pour la densité d'une loi normale multivariée
library(mvtnorm)
```

```r
naive_bayes <- function(index_test, index_train, matrix_logfreq)
{
  # Prédiction de la langue des textes du jeu de données test à partir du jeu d'apprentissage

  # Inputs :
  # "index_test" : vector, index des textes à classifier
  # "index_train" : vector, index des textes du jeu d'apprentissage
  # "matrix_logfreq" : matrix, matrice des log-fréquences de l'ensemble des textes

  # Output :
  # Liste des prédictions pour les textes du jeu de données test

  # Vecteur des prédictions
  y_pred <- c()

  # Estimation des paramètres sur le jeu d'apprentissage
  param <- compute_param(index_train, matrix_logfreq)

  # Calcul des probabilités marginales
  prob_en <- sum(matrix_logfreq[index_train, ncol(matrix_logfreq)]==1)/length(index_train)
  prob_fr <- 1-prob_en

  j <- 1

  # Boucle for sur le jeu de données test
  for(i in index_test)
  {
    # Calcul des probabilités sans division du dénominateur
    pi_en <- prob_en * dmvnorm(matrix_logfreq[i,-c(ncol(matrix_logfreq))], param$mu_en, param$var_en
```

```
    pi_fr <- prob_fr * dmvnorm(matrix_logfreq[i,-c(ncol(matrix_logfreq))], param$mu_fr, param$var_fr

    y_pred[j] <- ifelse(pi_en > pi_fr, 1, -1)

    j <- j + 1
  }

  return(y_pred)
}
```

2.3 Estimation des performances par validation croisée

Pour connaître les performances de notre classifieur, on utilise la validation croisée. On divise notre jeu de données en 5 folds, en faisant attention à stratifier sur la langue pour toujours avoir des textes anglais et français dans nos jeux d'apprentissage. La fonction `perf_nb` renvoie la matrice de confusion du classifieur.

```
# Import de splitTools pour la création des folds
library(splitTools)

# Import de caret pour la matrice de confusion
library(caret)

perf_nb <- function(matrix_logfreq)
{
  # Estimation des performances du classifieur de Bayes naïf par validation croisée

  # Input :
  # "matrix_logfreq" : matrix, matrice des log-fréquences de l'ensemble des textes
```

```r
  # Output :
  # Matrice de confusion

  # Création des folds
  kfolds <- create_folds(matrix_logfreq[, ncol(matrix_logfreq)], k=5, type="stratified")

  # Vecteur des langues
  y_theo <- c()

  # Vecteur des prédictions
  y_pred <- c()

  # Boucle for sur les folds
  for (fold in kfolds)
  {
    # Ajout des valeurs théoriques
    y_theo <- c(y_theo, matrix_logfreq[which(!(1:nrow(matrix_logfreq) %in% fold)), ncol(matrix_logfr

    # Ajout des prédictions
    y_pred <- c(y_pred,
                naive_bayes(which(!(1:nrow(matrix_logfreq) %in% fold)), fold, matrix_logfreq))
  }

  return(confusionMatrix(data=as.factor(y_pred),
                         reference = as.factor(y_theo)))

}

perf_nb(logf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction -1  1
##         -1 27  0
##         1   0 18
##
##                Accuracy : 1
##                  95% CI : (0.9213, 1)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 1.039e-10
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0
##             Specificity : 1.0
##          Pos Pred Value : 1.0
##          Neg Pred Value : 1.0
##              Prevalence : 0.6
##          Detection Rate : 0.6
##    Detection Prevalence : 0.6
##       Balanced Accuracy : 1.0
##
##        'Positive' Class : -1
##
```

Le classifieur de Bayes naïf a d'excellentes performances puisqu'il ne réalise aucune erreur. Il a su prédire parfaitement la langue pour tous les textes, et semble ainsi être un outil fiable et efficace.

3 - Classifieur markovien

3.1 Estimation des paramètres

On estime les paramètres de Markov (distribution initiale π et matrice de transition A) par estimateur du maximum de vraisemblance (MLE), on trouve alors :

- $\forall i \in$ {a, b,..., z}, $\pi_i = \frac{N_i}{\sum_{k=1}^{n} N_k}$
- $\forall j, i \in$ {a, b,..., z}, $A_{ji} = \frac{N_{ji}}{\sum_{k=1}^{n} N_{jk}}$

où N_{ji} est le nombre de passage de caractère j à i, et N_k est le nombre de caractère k.

Cependant, il est préférable d'estimer la distribution initiale π et la matrice de transition A par la quantité $\log(1 + \text{estimation})$, sinon la vraisemblance peut devenir très petite et être interprétée par 0 par la machine. Il serait alors impossible de comparer les vraisemblances pour chaque langue.

```
unigram <- function(dataset){
  # Unigram des caractères pour chaque langue

  # Input :
  # "dataset" : Dataset [Dataframe (X,Y)]

  # Output :
```

```r
  # Liste des unigrams (Tableau d'effectif de caractère) pour chaque langue

  X = table_of_dataset(dataset)
  n = dim(X)[2]

  # Unigram pour les textes français
  unigram_fr = as.vector(colSums(X[which(X[,n]==-1),-n]))
  names(unigram_fr) = colnames(X)[-n]

  # Unigram pour les textes anglais
  unigram_en = as.vector(colSums(X[which(X[,n]==1),-n]))
  names(unigram_en) = colnames(X)[-n]

  return(list(
    "fr" = unigram_fr,
    "en" = unigram_en
    ))
}

# Test :
# dataset_unigrams = unigram(dataset)
# print(dataset_unigrams)


normalize2 <- function(bigram, character_list){
  # Ajout des caractères manquants au bigram des caractères d'un texte

  # Inputs :
  # "bigram" : Bigram des caractères d'un texte
  # "names_character" : Ensemble des caractères possibles
```

```
  # Output :
  # "res" : bigram complet de chaque caractère (dans un texte)

  N = length(character_list)

  # Si aucune cellule ne manque, on retourne le tableau
  if ( (dim(bigram)[1]==N) && (dim(bigram)[2]==N) ) { return(bigram) }

  else{
    # Création d'une matrice nulle
    res = matrix(0,N,N)
    rownames(res) = character_list
    colnames(res) = character_list

    # Remplace les occurrences nulles par les valeurs du bigram
    for (letteri in rownames(bigram)){
      for (letterj in colnames(bigram)){
        res[letteri,letterj]=bigram[letteri,letterj]
      }
    }

  return (res)
  }
}

bigram <- function(dataset){
  # Bigram des caractères pour chaque langue

  # Input :
```

```r
# "dataset" : Dataset [Dataframe (X,Y)]

# Output :
# Liste des bigrams (Tableau d'effectif de passage de caractères) pour chaque langue

# On ajoute l'espace dans la liste des caractères
character_list = c(" ", "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o",
m = length(character_list)

# Création des matrices nulles
bigram_fr = matrix(0,m,m)
bigram_en = matrix(0,m,m)

n = length(dataset$x)

for (t in (1:n)) {
  text = as.vector(dataset$x[t])
  xt1 = strsplit(paste(" ",text, sep=""), split="") %>% unlist()
  xt2 = strsplit(paste(text, " ", sep=""), split="") %>% unlist()
  bigrams = table(xt1,xt2)

  # On ajoute les éventuelles cellules manquantes avec la fonction normalize2
  bigrams = normalize2(bigrams, character_list)
  if (dataset$y[t] == 1){bigram_en = bigram_en + bigrams}
  else{bigram_fr = bigram_fr + bigrams}
}

return(list(
  "fr" = bigram_fr,
  "en" = bigram_en
```

```
    ))
}

# Test :
# dataset_bigrams = bigram(dataset)
# print(dataset_bigrams)

normalize3 <- function(bigram, epsilon=10e-7){
  # Ajout d'epsilon aux valeurs nulles des bigrams + suppression transition espace-espace

  # Inputs :
  # "bigram" : bigram des caractères d'une langue
  # "epsilon" : quantité ajoutée aux valeurs nulles

  # Output :
  # "res" : bigram

  res = bigram

  for (letteri in rownames(res)){
    for (letterj in colnames(res)){
      # Remplace les occurrences nulles par epsilon
      if (res[letteri,letterj]==0){
        res[letteri,letterj] = epsilon
      }
    }
  }

  # On enlève les enchaînements espace-espace
  res[1,1] = 0
```

```r
  return (res)
}

estimation_Markov <- function(unigram, bigram){
  # Estimation des paramètres de markov pour une langue donnée

  # Input :
  # "dataset" : unigram & bigram pour une langue

  # Output : Liste des paramètres de Markov
  # "pi" : Distribution initiale
  # "A" : Matrice de transition

  # Préparation du bigram
  bigram = normalize3(bigram)

  pi = unigram / sum(unigram)
  A = bigram / rowSums(bigram)

  return(list(
    "distrib_init" = pi,
    "mat_transition" = A
    ))
}

dataset_unigrams = unigram(dataset)
dataset_bigrams = bigram(dataset)

Markov_FR = estimation_Markov(dataset_unigrams$fr, dataset_bigrams$fr)
```

```r
Markov_EN = estimation_Markov(dataset_unigrams$en, dataset_bigrams$en)

# Test :
# print(Markov_FR)
# print(Markov_EN)
```

3.2 MLE & Classifieur

```r
logprob_markov <- function(param, text_vector){
  # Probabilité "ajustée" (On additionne log(1+f)) d'un texte modélisé comme une chaine de markov de

  # Inputs :
  # "text_vector" : Vecteur de caractère
  # "param" : Paramètres de markov pi et A

  # Output : Langue estimée du texte (-1 ou 1)
  n = length(text_vector)
  res = log(1 + param$distrib_init[text_vector[1]])
  for (i in 2:n) {
    res = res + log(1 + param$mat_transition[text_vector[i-1],text_vector[i]])
  }
  return(res)
}

classifieur_Markovien <- function(text, Markov_FR, Markov_EN){
  # Prédiction de la langue d'un texte en le modélisant comme une chaîne de Markov

  # Inputs :
```

```r
  # "text" : Texte dont il faut prédire la langue (format : dataset$x[i])
  # "Markov_FR, Markov_EN" : Paramètres de markov pi et A pour chaque langue

  # Output : Prédiction de la langue du texte (-1 ou 1)

  # Transforme le texte en vecteur de caractère
  text_vector = strsplit(as.vector(text), split="") %>% unlist()

  # Si le texte commence par un espace, on le supprime
  if (text_vector[1] == " "){text_vector = text_vector[-1]}


  # Vraisemblance "ajustée" (on additionne log(1+f))
  ML_fr = logprob_markov(Markov_FR, text_vector)
  ML_en = logprob_markov(Markov_EN, text_vector)


  # Langue la plus probable selon les calculs
  if (ML_fr > ML_en){ return(-1) }
  else{ return(1) }
}

# Test
# for (i in 1:length(dataset$x)) {
#  cat(sprintf("Variable 'lang' prédite par le classifieur pour le texte %s : %s \n", #i, classifieu
#}
```

3.3 Validation croisée

```r
# Import de splitTools pour la création des folds
library(splitTools)

# Import de caret pour la matrice de confusion
library(caret)

# Kfold stratifié
kfolds = create_folds(dataset$y, k=5, type="stratified")

y_theo = c()
y_pred = c()
for (fold in kfolds) {

  # Partition entraînement/évaluation
  data_train = dataset[fold,]
  data_val = dataset[-fold,]

  # Apprentissage
  train_unigrams = unigram(data_train)
  train_bigrams = bigram(data_train)
  Markov_FR = estimation_Markov(train_unigrams$fr, train_bigrams$fr)
  Markov_EN = estimation_Markov(train_unigrams$en, train_bigrams$en)

  # Évaluation

  res = c()
  for (i in 1:length(data_val$x)) {
    res = c(res, classifieur_Markovien(data_val$x[i], Markov_FR, Markov_EN))
  }
```

```r
  # Ajout des valeurs théoriques
  y_theo = c(y_theo, data_val$y)
  # Ajout des prédictions
  y_pred = c(y_pred, res)

  #print(res)
  #print(data_val$y)
}
# Comparaison par matrice de confusion
confusionMatrix(data=as.factor(y_pred), reference = as.factor(y_theo))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction -1  1
##         -1 27  0
##         1   0 18
##
##                Accuracy : 1
##                  95% CI : (0.9213, 1)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 1.039e-10
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0
```

```
##             Specificity : 1.0
##          Pos Pred Value : 1.0
##          Neg Pred Value : 1.0
##              Prevalence : 0.6
##          Detection Rate : 0.6
##    Detection Prevalence : 0.6
##       Balanced Accuracy : 1.0
##
##        'Positive' Class : -1
##
```

4 - Décodage de langue par Viterbi

4.1 Création du texte aléatoire

On doit créer un court texte d'au plus 1 000 caractères enchaînant de manière aléatoire des phrases en français et en anglais. Tout d'abord, on importe les phrases de tous les textes sous forme d'un dataframe $(x_i, y_i)_{i=1,\dots,n}$ où $\forall i$, x_i est une phrase et y_i la langue de la phrase. Les phrases en anglais correspondent à $y = 1$, tandis que les phrases en français sont caractérisées par $y = -1$. La fonction `recup_sentences` s'inspire de la fonction `recup_data` et permet de créer le dataframe. Pour différencier les phrases, nous utilisons les signes de ponctuation suivants : point, point d'interrogation, point d'exclamation. Le dataset contient cependant quelques erreurs inévitables en raison des signes de ponctuation utilisés dans des cas précis, par exemple "M.de Balzac", "U.S.A", ou les onomatopées. Pour résoudre ce problème, le dataset ne comportera que des phrases d'au moins 20 caractères.

```
# Import de sbo pour utiliser la fonction tokenize_sentences
library(sbo)
```

```r
recup_sentences <- function(path = "./texts/"){
  # Même fonction que la fonction recup_data() mais pour chaque phrase

  # Input : chemin relatif (depuis le dossier où est stocké le .rmd) vers le dossier contenant les t

  # Output : dataframe décrit précédemment

  # Vecteur des langues des textes (les 18 premiers textes du dossier sont anglais, les 27 autres so
  vector_lang = c(rep(1,18),rep(-1,27))

  # Liste des phrases et de la langue associée
  sentences = c()
  lang_of_sentences = c()

  i = 1

  # Boucle for sur l'ensemble des fichiers .txt présents dans le dossier path
  for(t in list.files(path, pattern="txt")){
    text <- readLines(paste("./texts/",t, sep = ""), encoding = 'UTF-8') %>%
      paste(collapse=" ") %>%
      unlist %>%

      # Supprime les caractères spéciaux et les signes de ponctuation sauf "?!."
      str_remove_all("[,:;°-…—‘“/”'’\"-]") %>%
      str_remove_all("\\(") %>%
      str_remove_all("\\)") %>%
      str_remove_all("\\«") %>%
      str_remove_all("\\»") %>%
      str_remove_all("\\[") %>%
```

```
      str_remove_all("\\]") %>%

      # Retire les chiffres
      removeNumbers() %>%

       # Transforme les majuscules en minuscules
      str_to_lower() %>%

      # Remplace les lettres accentuées
      chartr(old = "àáâäãèéêëìíîïòóôöõùúûüýÿçñœæ", new = 'aaaaaeeeeiiiioooooouuuuyycnea')

      # Sépare les phrases du texte
      sentences_by_text = tokenize_sentences(as.vector(text))

      # Stocke les nouvelles phrases
      sentences = c(sentences, sentences_by_text)

      # Stocke la langue des nouvelles phrases = langue du texte
      lang_of_sentences = c(lang_of_sentences,
                            rep(vector_lang[i],length(sentences_by_text))
                            )
      i = i+1
  }

  return(data.frame(
    x =  sentences,
    y = lang_of_sentences
  ))
}
```

```r
# Création du dataset avec les phrases
dataset_sentences = recup_sentences()

# Le dataset ne comportera que des phrases d'au moins 20 caractères.
dataset_sentences = dataset_sentences[which(nchar(as.vector(dataset_sentences$x))>=20),]

# Aperçu & Dimension
head(dataset_sentences)
```

```
##
## 2
## 3 from forth the fatal loins of these two foes a pair of starcrossed lovers take their life whose
## 4
## 5
## 6
## 7
##   y
## 2 1
## 3 1
## 4 1
## 5 1
## 6 1
## 7 1
```

```r
dim(dataset_sentences)
```

```
## [1] 896   2
```

Une fois que le dataset contenant les phrases est créé, on souhaite générer un texte enchaînant de manière aléatoire des phrases en français et en anglais. Pour générer la première phrase, on choisit par défaut des probabilités de tirage égales aux proportions initiales renvoyées par la fonction `initial_distrib`. Cette fonction calcule la proportion de phrases en anglais et en français dans le dataset pour en déduire les probabilités initiales. La fonction `texte_made1` génère ensuite une suite de phrases composant le texte.

```
initial_distrib <- function(dataset_sentences){
  # Détermine la distribution initiale (proportions de phrases en anglais et en français)

  # Input :
  # "dataset_sentences" : Dataframe des phrases récupérées et langues associées

  # Output : distribution initiale

  N = length(dataset_sentences$y)

  # Proportions de phrases en français et en anglais dans le dataset
  N_fr = length(which(dataset_sentences$y == -1))
  N_en = N - N_fr

  # Distribution initiale
  pi = c(N_fr, N_en)/N

  return(pi)
}
```

```r
text_made1 <- function(pi, mat_transition, dataset_sentences){
  # Génération d'un texte constitué de phrases anglaises et françaises tirées aléatoirement

  # Inputs :
  # "dataset_sentences" : Dataframe des phrases récupérées et langues associées
  # "pi, mat_transition" : Paramètres de Markov

  # Output : Dataframe de même format que l'input, avec les phrases du texte créé

  # Détermine les index des phrases en français et en anglais
  indx_fr = which(dataset_sentences$y == -1)
  indx_en = which(dataset_sentences$y == 1)

  # Index déjà utilisés pour chaque langue (phrases déjà sélectionnées)
  indx_fr_taken = c()
  indx_en_taken = c()

  # Phrase initiale
  # Etat 1 : fr (français), état 2 : en (anglais)
  # Tire aléatoirement un état avec probabilité pi
  Z = sample.int(2, 1, prob=pi)

  # Si la première phrase est en français
  if (Z==1){
    new_indx = sample(indx_fr, 1)
    indx_fr_taken = c(indx_fr_taken,new_indx)
  }

  # Si la première phrase est en anglais
  else{
```

```r
  new_indx = sample(indx_en, 1)
  indx_en_taken = c(indx_en_taken,new_indx)
}

# Stocke le premier index
indx.res = new_indx

# Nombre de caractères du texte après ajout de la phrase
len = nchar(as.vector(dataset_sentences$x[new_indx]))

# Tant que le texte créé ne fait pas plus de 1 000 caractères
while(len<=1000){
  # Tire aléatoirement un état avec probabilité définie par mat_transition
  # Etat 1 : fr (français), état 2 : en (anglais)
  Z = sample.int(2, 1, prob=mat_transition[Z,])

  # Si la phrase est en français
  if (Z==1){

    # S'il n'y a pas encore eu de phrases en français
    if (length(indx_fr_taken)==0){ new_indx = sample(indx_fr, 1) }

    # S'il y a déjà eu des phrases en français
    else{ new_indx = sample(indx_fr[-indx_fr_taken], 1) }

    indx_fr_taken = c(indx_fr_taken,new_indx)
  }

  # Si la phrase est en anglais
  else{
```

```r
      # S'il n'y a pas encore eu de phrases en anglais
      if (length(indx_en_taken)==0){ new_indx = sample(indx_en, 1) }

      # S'il y a déjà eu des phrases en anglais
      else{ new_indx = sample(indx_en[-indx_en_taken], 1) }
      indx_en_taken = c(indx_en_taken,new_indx)
    }

    # Stocke le nouvel index
    indx.res = c(indx.res,new_indx)

    # Nombre de caractères du texte après ajout de la phrase
    len = len + nchar(as.vector(dataset_sentences$x[new_indx]))

    #print(len)
    #print(new_indx)
    #print(indx_fr_taken)
    #print(indx_en_taken)
    #print(Z)
    #print(as.vector(dataset_sentences$x[new_indx]))
  }

  # On supprime la dernière phrase pour avoir un texte final de moins de 1 000 caractères
  indx.res = indx.res[-length(indx.res)]

  # Sélection des phrases & de la langue associée
  return(dataset_sentences[indx.res,])
}
```

```
pi = initial_distrib(dataset_sentences)
A = matrix(c(0.2,0.5,0.8,0.5),2)

new_dataset = text_made1(pi, A, dataset_sentences)
head(new_dataset)
```

```
##
## 571 et comme pour bercer les lenteurs de la route je chanterai des airs ingenus je me dis quelle r
## 202
## 987
## 277
## 230
## 895
##      y
## 571 -1
## 202  1
## 987 -1
## 277  1
## 230  1
## 895 -1
```

4.2 Algorithme de Viterbi

Dans cette partie, nous considérons un modèle de Markov caché où les variables observées $X = (x_1, x_2, \ldots, x_T)$ sont des phrases et les états cachés sont les langues possibles dans lesquelles sont écrites les phrases $S = (s_1 = fr, s_2 = en)$. En ce qui concerne les probabilités d'émission, on modélise les observations (phrases) par des chaînes de Markov de caractères dont les paramètres (distribution initiale et

matrice de transition) dépendent de la langue (état caché). Les T phrases générées à la question précédentes sont notées $O = o_1, o_2, \ldots, o_T$. Elles sont alors définies comme des modalités que peut prendre la variable X, de telle manière que $x_t = i$ si la phrase t-ième est o_i.

Pour les variables cachées, on décide nous-même des paramètres de Markov qui permettent de choisir dans quelle langue nous allons tirer chaque phrase. (On peut choisir une distribution initiale en fonction des proportions). C'est de cette manière que l'on génère les observations dans la question 1.

L'objectif de la méthode Viterbi appliqué à cet exercice est de déterminer les langues dans lesquelles sont écrites chacune des phrases mélangées, c'est-à-dire de déterminer les états cachés $Z = z_1, z_2, \ldots, z_T$ associés à $X = (x_1, x_2, \ldots, x_T)$

Pour rappel, on continue à utiliser la quantité $\log(1 + \text{estimation})$, sinon la vraisemblance peut devenir très petite et être interprétée par 0 par la machine.

```
logprobmarkov_forapply <- function(param, sentence){
  # Même fonction que logprob_markov mais avec comme argument une phrase qui n'est pas sous forme de

  # phrase sous forme de vecteur de caractère
  vector_sentence = strsplit(sentence, split="") %>% unlist()

  # logprob_markov
  res = logprob_markov(param, vector_sentence)
  return(res)
}

build_emission <- function(dataset, MarkovFR, MarkovEN){
  vector_sentences = as.vector(dataset)
```

```r
  # On applique (sapply) logprob_markov (probabilité d'avoir telle phrase sachant telle langue) à ch
  emission_fr = sapply(vector_sentences,
                       FUN = function(x) logprobmarkov_forapply(MarkovFR,x)
                       )
  emission_en = sapply(vector_sentences,
                       FUN = function(x) logprobmarkov_forapply(MarkovEN,x)
                       )

  res = rbind(as.vector(emission_fr), as.vector(emission_en))
  # Normalisation pour que la somme des lignes fasse 1
  res = res/rowSums(res)
  return(t(res))
}

# Test
# print(build_emission(new_dataset$x, Markov_FR, Markov_EN))
```

```r
viterbi_sentences <- function(Pi, A, X, MarkovFR, MarkovEN){
  # Algorithme de Viterbi pour trouver les passages en français et en anglais du texte créé

  # Inputs :
  # "Pi, A" : Paramètres de Markov pour les états cachés (distribution initiale et matrice de transi
  # "X" : Observations : colonne de dataframe stockant T phrases et leurs langues associées (X[i,] l
  # "Markov_FR, Markov_EN" : Paramètres d'émission (pi et A) pour chaque état caché possible (langue

  # Output : États cachés Z (langues : fr=1, en=2)
```

```
# Création de la matrice d'émission : on regarde X comme une variable à T modalités (T phrases obs
B = build_emission(X, MarkovFR, MarkovEN)
# print(B)

# Algoritme de Viterbi
# rq : on aditionne B[t,l] car B contient déjà les logarithmes

K = dim(A)[1] # K = 2 (fr, en)
T = length(X)
Z = rep(0,T)
S = matrix(rep(0, T*K),T,K)
logV = matrix(rep(-Inf, T*K),T,K)
logV[1,] = log(pi) + B[1,] # B est déjà une logprobabilité, pas besoin de faire logB, juste additi
#print(logV)

# Forward
for (t in 2:T) { # t >=2
  for (l in 1:K) {
    logV[t,l] = max(logV[t-1,] + log(1+A[,l]) + B[t,l]) # B est déjà une logprobabilité, pas besoi
    S[t-1,l] = which.max (logV[t-1,] + log(1+A[,l]) + B[t,l]) # B est déjà une logprobabilité, pas
    #print(logV)
    #print(S)
  }
}

# Backward
Z[T] = which.max(logV[T,])
for (t in (T-1):1) {
  Z[t] = S[t,Z[t+1]]
}
```

```r
  return(Z)
}

viterbi_result1 = viterbi_sentences(pi, A, new_dataset$x, Markov_FR, Markov_EN)
```

```r
viterbi_result1
```

```
##  [1] 1 2 1 2 2 1 2 1 2 1 2
```

```r
# Vérification : Transformer les états (1,2) en variables y (-1,1)
viterbi_result1 = 2*viterbi_result1 - 3

# Matrice de confusion & scoring
confusionMatrix(data=as.factor(viterbi_result1), reference = as.factor(new_dataset$y))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction -1 1
##         -1  4 1
##         1   0 6
##
##                Accuracy : 0.9091
##                  95% CI : (0.5872, 0.9977)
##     No Information Rate : 0.6364
##     P-Value [Acc > NIR] : 0.05049
```

```
##
##                  Kappa : 0.8136
##
##  Mcnemar's Test P-Value : 1.00000
##
##             Sensitivity : 1.0000
##             Specificity : 0.8571
##          Pos Pred Value : 0.8000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.3636
##          Detection Rate : 0.3636
##    Detection Prevalence : 0.4545
##       Balanced Accuracy : 0.9286
##
##        'Positive' Class : -1
##
```

5 Exercice 4bis - Décodage de langue par Viterbi

Cette variante de notre exercice 4 est celle réellement demandée : Créer un texte d'au plus 1000 caractère et repérer les passage en anglais et français à l'aide de l'algorithme de Viterbi, en modélisant cette fois une **chaine de markov cachee pour chaque caractere**

5.1 Textes fabriqués

On résutilise le même procédé qu'avant en prenant une probabilité uniforme de prendre des passages anglais ou français. Cela revient à créer un texte manuellement.

```r
text_made2 <- function(dataset_sentences){
  # Génération d'un texte constitué de phrases anglaises et françaises tirées aléatoirement

  # Inputs :
  # "dataset_sentences" : Dataframe de phrases récupérées et langues associées

  # Output : vecteur de caractère + vecteur des états pour chaque caractère (1 et 2)

  text.res = c()
  lang.res = c()

  # Détermine les index français et anglais
  indx_fr = which(dataset_sentences$y == -1)
  indx_en = which(dataset_sentences$y == 1)

  # Index déjà pris pour chaque langue (phrases déjà sélectionnées)
  indx_fr_taken = c()
  indx_en_taken = c()

  # Nombre de caractères du texte après ajout de la phrase
  len = 0

  # Tant que le texte créé ne fait pas plus de 1 000 caractères
  while(len<=1000){
    Z = sample.int(2, 1) # etat 1 : fr, etat 2 : en
    if (Z==1){
      if (length(indx_fr_taken)==0){ new_indx = sample(indx_fr, 1) }
      else{ new_indx = sample(indx_fr[-indx_fr_taken], 1) }
      indx_fr_taken = c(indx_fr_taken,new_indx)
```

```r
    }
    else{
      if (length(indx_en_taken)==0){ new_indx = sample(indx_en, 1) }
      else{ new_indx = sample(indx_en[-indx_en_taken], 1) }
      indx_en_taken = c(indx_en_taken,new_indx)
    }
    new_sentence = dataset_sentences$x[new_indx] %>% as.vector()

    # Nombre de caractères du texte après ajout de la phrase
    len = len + nchar(new_sentence)

    # Nouveau passage (vecteur de caractères)
    new_sentence = strsplit(new_sentence, split = " ") %>% unlist()
    new_sentence = paste(new_sentence, sep="")
    new_sentence = strsplit(new_sentence, split = "") %>% unlist()


    # Ajout aux résultats (vecteur de caractères et d'états)
    text.res = c(text.res, new_sentence)
    lang.res = c(lang.res, rep(Z, length(new_sentence)))
  }

  # Sélection des phrases & de la langue associée
  return(list(
    "text" = text.res,
    "lang" = lang.res))
}
manual_text = text_made2(dataset_sentences)
# Aperçu
print(manual_text$text)
```

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

```
## [523] "n" "t" "h" "e" "s" "a" "m" "e" "j" "o" "b" "b" "e" "f" "o" "r" "e" "t"
## [541] "h" "a" "t" "n" "o" "u" "s" "n" "a" "v" "o" "n" "s" "p" "a" "s" "e" "n"
## [559] "c" "o" "r" "e" "a" "s" "s" "e" "z" "d" "e" "l" "e" "m" "e" "n" "t" "s"
## [577] "p" "o" "u" "r" "n" "o" "u" "s" "p" "r" "o" "n" "o" "n" "c" "e" "r" "s"
## [595] "u" "r" "l" "e" "s" "t" "e" "m" "p" "e" "r" "a" "t" "u" "r" "e" "s" "h"
## [613] "i" "v" "e" "r" "n" "a" "l" "e" "s" "a" "v" "e" "n" "i" "r" "u" "n" "g"
## [631] "r" "a" "n" "d" "f" "e" "u" "t" "r" "e" "a" "l" "o" "n" "g" "u" "e" "p"
## [649] "l" "u" "m" "e" "o" "m" "b" "r" "a" "i" "t" "s" "o" "n" "e" "i" "l" "q"
## [667] "u" "i" "s" "a" "l" "l" "u" "m" "e" "e" "t" "s" "e" "t" "e" "i" "n" "t"
## [685] "m" "a" "r" "t" "i" "n" "e" "z" "b" "l" "e" "s" "s" "e" "u" "n" "e" "d"
## [703] "e" "f" "a" "i" "t" "e" "a" "s" "s" "u" "m" "e" "e" "f" "r" "i" "a" "r"
## [721] "l" "a" "u" "r" "e" "n" "c" "e" "h" "o" "l" "d" "d" "a" "u" "g" "h" "t"
## [739] "e" "r" "i" "d" "o" "s" "p" "y" "a" "k" "i" "n" "d" "o" "f" "h" "o" "p"
## [757] "e" "w" "h" "i" "c" "h" "c" "r" "a" "v" "e" "s" "a" "s" "d" "e" "s" "p"
## [775] "e" "r" "a" "t" "e" "a" "n" "e" "x" "e" "c" "u" "t" "i" "o" "n" "a" "s"
## [793] "t" "h" "a" "t" "i" "s" "d" "e" "s" "p" "e" "r" "a" "t" "e" "w" "h" "i"
## [811] "c" "h" "w" "e" "w" "o" "u" "l" "d" "p" "r" "e" "v" "e" "n" "t"
```

```
length(manual_text$text)
```

```
## [1] 826
```

```
length(manual_text$lang)
```

```
## [1] 826
```

5.2 Chaine de Markov cachée & Emission

On prépare les paramètres pour modéliser la chaine de Markov cachée : Matrice de Transition, Distribution initiale et Matrice d'emission.

```
# Déterminer les paramètres de la chaine de Markov cachée
alpha = 0.9
pi = rep(1/2,2)
markov_param = list(
  "init_distrib" = pi,
  "mat_transition" = matrix(c(alpha, 1-alpha, 1-alpha, alpha), 2)
  )

print(markov_param)
```

```
## $init_distrib
## [1] 0.5 0.5
##
## $mat_transition
##      [,1] [,2]
## [1,]  0.9  0.1
## [2,]  0.1  0.9
```

```
# Matrice d'emission à partir des fréquence de que caractère pour le français et l'anglais
emission_fr = dataset_unigrams$fr / sum(dataset_unigrams$fr)
emission_en = dataset_unigrams$en / sum(dataset_unigrams$en)
emission = cbind(emission_fr, emission_en)
```

```r
rownames(emission) = c("a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p
print(emission)
```

```
##    emission_fr emission_en
## a 8.129678e-02 0.082334954
## b 9.648409e-03 0.017920698
## c 3.148637e-02 0.027595368
## d 3.646940e-02 0.040453156
## e 1.724008e-01 0.124918542
## f 1.030355e-02 0.022833225
## g 9.926347e-03 0.021655221
## h 8.695480e-03 0.052784601
## i 6.912708e-02 0.069602486
## j 4.983026e-03 0.001553963
## k 1.588215e-04 0.008496666
## l 5.483314e-02 0.043410697
## m 3.114888e-02 0.025690511
## n 7.121161e-02 0.073036242
## o 5.592504e-02 0.073437265
## p 2.765480e-02 0.019549852
## q 1.349983e-02 0.000852173
## r 6.734033e-02 0.061431651
## s 8.429453e-02 0.067096095
## t 6.736019e-02 0.087498120
## u 6.841238e-02 0.028096646
## v 1.625936e-02 0.010852674
## w 9.926347e-05 0.018973382
## x 4.030097e-03 0.001729410
## y 2.263207e-03 0.017920698
```

```
## z 1.171309e-03 0.000275703
```

5.3 Algorithme de Viterbi

```
Viterbi_caractere <- function(Pi,A,B,X){
  # Algorithme de Viterbi pour trouver les passages en français et en anglais du texte créé

  # Inputs :
  # "Pi, A" : Paramètres de Markov pour les états cachés (distribution initiale et matrice de transi
  # "X" : Observations : vecteur de caractères
  # "B" : Matrice d'emission

  # Output : États cachés Z (langues : fr=1, en=2)

  K = dim(A)[1] # K = 2 (fr, en)
  T = length(X)
  Z = rep(0,T)
  S = matrix(rep(0, T*K),T,K)
  logV = matrix(rep(-Inf, T*K),T,K)
  logV[1,] = log(pi*B[X[1],])
  #print(logV)

  # Forward
  for (t in 2:T) { # t >=2
    for (l in 1:K) {
      logV[t,l] = max(logV[t-1,] + log(A[,l]) + log(B[X[t],l]))
      S[t-1,l] = which.max (logV[t-1,] + log(A[,l]) + log(B[X[t],l]))
      #print(logV)
```

```
      #print(S)
    }
  }

  # Backward
  Z[T] = which.max(logV[T,])
  for (t in (T-1):1) {
    Z[t] = S[t,Z[t+1]]
  }

  return(Z)
}
viterbi_result2 = Viterbi_caractere(markov_param$init_distrib, markov_param$mat_transition, emission
print(viterbi_result2)
```

```
##   [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 2
##  [38] 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 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
## [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
## [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
## [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
## [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
## [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
## [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
## [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
## [371] 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 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
## [445] 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
## [482] 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
```

```
## [519] 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 1 1 1 1 1 1 1 1 1
## [556] 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
## [593] 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
## [630] 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
## [667] 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
## [704] 1 1 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
## [741] 2 2 2 2 2 2 2 2 2 2 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
## [778] 1 1 1 1 1 1 1 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
## [815] 2 2 1 1 1 1 1 1 1 1 1 1
```

```r
# Matrice de confusion & scoring
confusionMatrix(data=as.factor(viterbi_result2), reference = as.factor(manual_text$lang))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 498  58
##          2   2 268
##
##                Accuracy : 0.9274
##                  95% CI : (0.9075, 0.9441)
##     No Information Rate : 0.6053
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8433
##
##  Mcnemar's Test P-Value : 1.243e-12
```

```
##
##             Sensitivity : 0.9960
##             Specificity : 0.8221
##          Pos Pred Value : 0.8957
##          Neg Pred Value : 0.9926
##              Prevalence : 0.6053
##          Detection Rate : 0.6029
##    Detection Prevalence : 0.6731
##       Balanced Accuracy : 0.9090
##
##        'Positive' Class : 1
##
```

En guise de commentaire, on peut dire qu'on trouve une accuracy relativement bonne (0.97 au premier essai).