Saisie Prédictive

Aloys Aymrick

2022-07-25

```
                                Saisie Prédictive
```

```
library(NLP) ##Natural Language Processing
library(tm) ##Pour le traitement de du dictionaire
```

```
## Warning: package 'tm' was built under R version 4.3.2
```

Preparation du dictionnaire pour les n-grams

1. Charger la bibliotheque

Chargement de la bibliotheque dans un vcorpus du package tm

```
Data1 <- VCorpus(DirSource("C:/Users/aymer/OneDrive/Desktop/ESSFAR/Projet ESSFAR/R/Projet R 202
2/App/Bibliotheque"))
```

Convertir en minuscule

```
Data2 <- tm_map(Data1, content_transformer(tolower))
```

Enlever les ponctuation

```
Data3 <- tm_map(Data2, removePunctuation)
```

###Maintenant notre dictionaire est pret pour la modalisation en n-grams

###Pour proceder nous allons utiliser le package “Rweka” qui neccesite un environnement java pour pouvoir etre execute

2. Transformation en Bi-gram

```
library("RWeka")
```

```
## Warning: package 'RWeka' was built under R version 4.3.2
```

Convertir le dictionnaire en Bigram

##Creation d’une fonction bigram tokenizer qui convertira une chaine en bigram a l’aide de la fonction weka_control disponible sur le package RWeka

```
#Creation d'une fonction bigram tokenizer qui convertira une chaine en bigram a l'aide de la fon
ction weka_control disponible sur le package RWeka

BigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))
```

##Nous appliquons maintenant la fonction BigramTokenizer sur l'ensemble des phrases dans notre corpus et nous conertissons les résultats en une matrice DocumentTerm

3. Frequence de chaque token

```
#Nous appliquons maintenant la fonction BigramTokenizer sur l'ensemble des phrases dans notre co
rpus et nous convertissons les résultats en une matrice DocumentTerm

Data_bigrams <- DocumentTermMatrix(Data3, control = list(tokenize = BigramTokenizer))
inspect(Data_bigrams)
```

```
## <<DocumentTermMatrix (documents: 1, terms: 44014)>>
## Non-/sparse entries: 44014/0
## Sparsity           : 0%
## Maximal term length: 61
## Weighting          : term frequency (tf)
## Sample             :
##           Terms
## Docs       are you have a i am if you in the to be want to will be you are
##   spam.csv     144    124   93     81     97    83      80      81     100
##           Terms
## Docs       you have
##   spam.csv       84
```

```
##Comme nous pouvons le remarquer les colonnes contient des bigrams et leurs frequence
```

```
bigram_frequency <- sort(colSums(as.matrix(Data_bigrams)),
                         decreasing=TRUE)

##Maintenant nous avons la frequence de chaque bigram en ordre decroissant en utilisant la fonct
ion sort
```

```
#Nous convertissons le tableau des fréquence en data frame pour pouvoir faciliter l'analyse

bigram_df <- data.frame(bigrams=names(bigram_frequency),
                        freq=bigram_frequency)
head(bigram_df)
```

	bigrams <chr>	**freq** <dbl>
are you	are you	144
have a	have a	124

	bigrams <chr>	freq <dbl>
you are	you are	100
in the	in the	97
i am	i am	93
you have	you have	84

6 rows

Maintenant, nous allons créér une base de données bigram avec ce data frame

```
#Ici pour chaque ligne nous allons séparer les bigram en premier et deuxieme mot puis le stocker dans le meme data frame

for ( i in 1:nrow(bigram_df))
  {
    grams = unlist(strsplit(as.character(bigram_df$bigrams[i])," "))

  bigram_df$first[i]= grams[1]
  bigram_df$second[i]= grams[2]
}
bigram_df[1:10,]
```

	bigrams <chr>	freq <dbl>	first <chr>	second <chr>
are you	are you	144	are	you
have a	have a	124	have	a
you are	you are	100	you	are
in the	in the	97	in	the
i am	i am	93	i	am
you have	you have	84	you	have
to be	to be	83	to	be
if you	if you	81	if	you
will be	will be	81	will	be
want to	want to	80	want	to

1-10 of 10 rows

Testons notre bigram

```
#Nous allons prédire le qui suit apres "back"

#Nous allons filtrer le dataframe ou le premier mot est "back"
mot_filtre = bigram_df[
  bigram_df$first == "back",
  c("freq", "second")]

#Ordre de frequence decroissante
prochain_mot = mot_filtre[
  with(mot_filtre, order(-freq)), ]

#Le prochaine mot prédit
(prochain_mot$second)
```

```
##  [1] "to"         "in"         "on"         "from"       "if"
##  [6] "at"         "and"        "help"       "home"       "my"
## [11] "now"        "so"         "til"        "1u"         "by"
## [16] "soon"       "ull"        "ur"         "2"          "4"
## [21] "a"          "after"      "again"      "amp"        "any"
## [26] "because"    "before"     "bit"        "can"        "chat"
## [31] "come"       "did"        "do"         "e5d4morrow" "earlier"
## [36] "every"      "excellent"  "for"        "fr"         "g"
## [41] "good"       "half"       "have"       "i"          "id"
## [46] "jess"       "later"      "lemme"      "like"       "loads"
## [51] "log"        "monday"     "n"          "name"       "next"
## [56] "or"         "our"        "pain"       "sch"        "take"
## [61] "that"       "the"        "then"       "there"      "though"
## [66] "tomo"       "tonight"    "urself"     "we"         "when"
## [71] "with"       "xafter"     "your"
```

```
#Donc nous avons la liste des mots les plus probables apres le mot back
```