Projet en Traitement des données massives (S. Boucheron)

Minh Tuan Do, Etienne Kintzler

Introduction

Le projet a pour objectif d'appliquer deux méthodes, le Locally Sensitive Hashing et la Latent Dirichlet Allocation à une base de donnée célèbre, les mails de l'entreprise Américaine Enron.

Les languages de programmation utilisés sont Python pour une des tâches de pré-traitement et R pour le reste du pré-traitement ainsi que pour l'application des méthodes vues en cours. L'ensemble des codes et ressources utilisées sont disponibles à l'adresse suivante :
https://github.com/etiennekintzler/Donnees-Massives-Boucheron/

Description de la base de donnée Enron

La base de donnée est composée d'environ 500 000 messages envoyées/reçues par environ 150 utilisateurs (principalement des managers). La base de données est disponible à l'adresse suivante :
https://www.cs.cmu.edu/~./enron/

Pré-traitement

Nous utilisons un script python (cf annexe 1) afin de sélectionner les mails des sous-dossier "sent" uniquement. Le script permet aussi de supprimer les élements propre à la structure des mails comme :"To","From","Subject:" etc. Du fait du nombre particulièrement important du corpus (environ 130 000 fichiers textes) nous avons réaliser un tirage aléatoire selon $\mathcal{B}(1, 0.3)$ afin de réduire la taille de l'échantillon

1. Locally Sensitive Hashing

Perform approximate nearest neighbor analysis

Introduction

Dans le context de text-mining, si l'on veut détecter des documents similaires ou des documents identiques, on doit construire un modèle qui compare la similarité entre tous les paires de documents. Par exemple, pour le problème de construction de clusters ou k-plus proches voisins, on peut définir une distance de similarité de Jaccard entre deux documents, et appliquer des algorithmes d'apprentissage non-supervisés. Cependent, le coût de calcul va exploser rapidement lorsque la taille des données ou la dimension du problème augmente. Pour remédier à ce problème, on peut utiliser l'algorithme LSH qui utilise des fonctions min-hashing randomisées pour estimer le cofficient de similarité de Jaccard entre des documents afin de trouver des paires de documents tels que leur coefficient de similarité est supérieur à un niveau donné. L'idée principale est d'utiliser une famille de fonction de hachage choisies telles que des points proches dans l'espace d'origine aient une forte probabilité d'avoir la même valeur de hachage.

Modèle LSH pour text mining

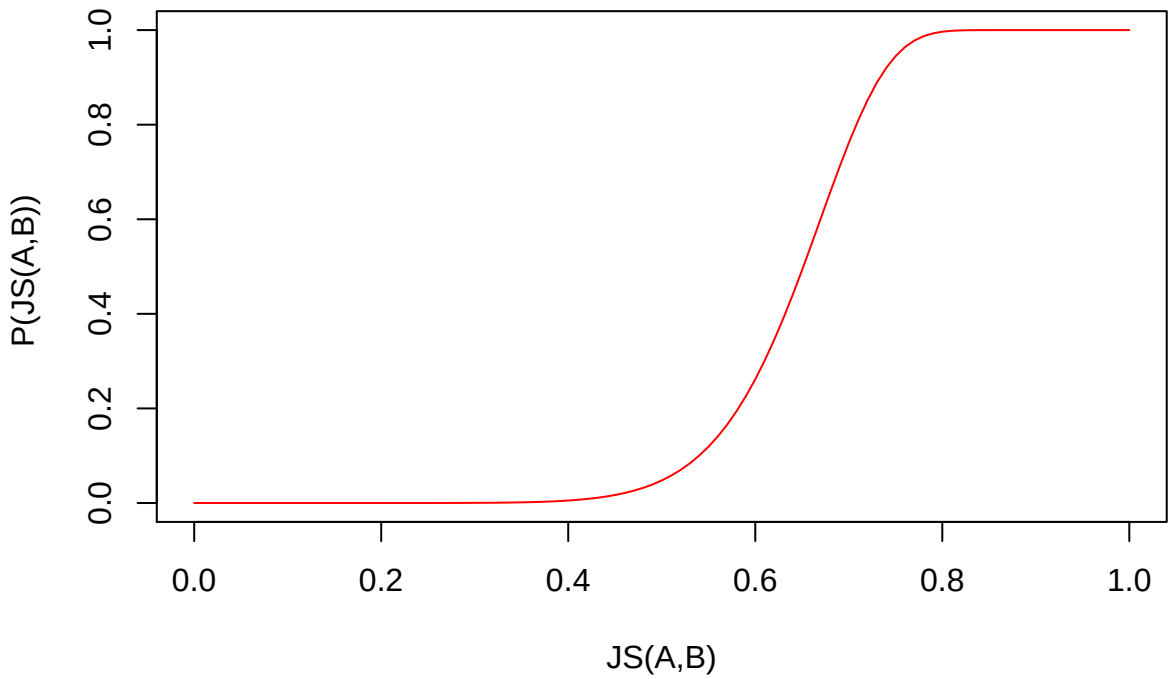
Fonction min-hashing

La définition de la similarité-Jaccard entre deux documents A et B:
$JS(A,B) = \frac{|A \bigcap B|}{|A \bigcup B|}$
La fonction min-hashing $m(.)$ va rendre le rang de premier terme appartient au document selon l'ordre généré par cette fonction hashing. On définit une variable aléatoire $x(A,B)$ telle que :
$x(A,B) = 1$ si $m(A) = m(B)$
$x(A,B) = 0$ sinon.
Par conséquence, la variable aléatoire $x(A,B)$ suit la loi B(JS(A,B)). Donc, on peut estimer la similarité de Jaccard, en générant des observations iid de cette loi $B(JS(A,B))$ par des fonctions min-hashing et moyennant des valeurs.

Locality sensitive hashing (LSH)

Dans le cardre de text mining, on utilise la fonction hashing par bloc des signatures générées par les fonctions min-hashing. Effectivement, si l'on considère n signatures avec b blocs de taille r ou bien : $n = b * r$, la fonction LSH considère que deux documents forment un paire de candidats s'ils coïncident au moins pour un bloc. La probabilité telle que une paire de documents est une paire de candidat est ainsi donnée par :
$P(JS(A,B)) = 1 - (1 - JS(A,B)^r)^b$

```
plot(seq(0,1,0.01),   1- ( 1 -  seq(0,1,0.01)**10 )**50 , type = "l",
     xlab = "JS(A,B)", ylab = "P(JS(A,B)) ", col="red" )
```



Cette probabilité possède une forme de courbe en S, croissante $P(0) = 0$ et $P(1) = 1$. Cette probabilité nous garantit la bonne définition d'une fonction de hachage, c'est-à-dire que les fonctions de hachage doivent permettre aux points proches d'entrer fréquemment en collision (i.e. $h(p) = h(q)$) et inversement, les points éloignés ne doivent que rarement entrer en collision.

On peut ajuster les valeurs de b et r pour manipuler le seuil de $JS(A,B)$ telles que A et B sont une paire de candidats avec une grande probabilité.

Implémentation sous R

```
plyr::laply(.data=c("ggplot2","tm","topicmodels","filehash",
                    "SnowballC","slam","textreuse","dplyr","magrittr","foreach"
                    ,"igraph"),
            .fun=require,
            character.only=TRUE,
            quietly=TRUE,
            warn.conflicts=FALSE)

#Repertoire dans lequel se trouve le dossier avec les mails
setwd("~/R/projet_boucheron/")
enronpath <- paste(getwd(),"DATA",sep="/") # dir of enron data
```

La fonction suivante permets de calculer le seuil de Jaccard avec des valeurs de h et b données. Donc, on chosit des valeurs de h et b telles que le seuil soit convenable (entre 0.5 et 0.7)

```
lsh_threshold(h = 200, b = 25)
```

```
## [1] 0.6687403
```

Générer des fonctions min-hashing

```
minhash <- minhash_generator(n = 200)
```

Construire le corpus de TextReuseCorpus pour utiliser LSH

```
dir <- enronpath

Corpus <- VCorpus(DirSource(dir, encoding= "UTF-8"),
                  readerControl=list(language= "en") )

Corpus <- Corpus %>%
        tm_map(content_transformer(tolower),lazy=TRUE) %>%
        tm_map(removeWords,stopwords("english"), lazy=TRUE) %>%
        tm_map(stemDocument,lazy=TRUE) %>%
        tm_map(removePunctuation,lazy=TRUE) %>%
        tm_map(stripWhitespace,lazy=TRUE) %>%
        tm_map(removeNumbers,lazy=TRUE)
```

```
barCorpus <- foreach (doc=Corpus, .combine="c") %do% TextReuseCorpus(text=doc$content,
                  tokenizer = tokenize_words,
                  progress = TRUE,
                  keep_tokens = TRUE,
                  keep_text = TRUE,
                  skip_short = TRUE)
```

```
dir    <- enronpath
corpus <- TextReuseCorpus(dir = dir , minhash_func = minhash, skip_short = TRUE,
                          n=10)
```

Construire les buckets. Le nombre de bandes est divisé par le nombre de fonctions min-hashing.

```
####CONSTRUIRE LES BUCKETS

buckets <- lsh(corpus, bands = 25, progress = FALSE)
```

Trouver de paires de candidats liées à un document particulier.

```
baxter_matches <- lsh_query(buckets, "100")
baxter_matches
```

```
## Source: local data frame [1 x 3]
##
##       a     b score
##   (chr) (chr) (dbl)
## 1   100   637    NA
```

Trouver tous les paires de candidats

```
#returns all potential pairs of matches.
candidates <- lsh_candidates(buckets)
candidates
```

```
## Source: local data frame [7,913 x 3]
##
##         a      b score
##     (chr)  (chr) (dbl)
## 1     100    637    NA
## 2  100015 100022    NA
## 3  100029  99562    NA
## 4  100041  82320    NA
## 5   10005  11749    NA
## 6  100065  97494    NA
## 7  100109 114224    NA
## 8  100109  96451    NA
## 9  100109  97906    NA
## 10 100109  99888    NA
## ..    ...    ...   ...
```

Comparaison les paires de candidats avec la similarité de Jaccard

```
#lsh_compare() to apply a similarity function to the candidate pairs of documents
lsh_compare(candidates, corpus, jaccard_similarity, progress = FALSE)
```

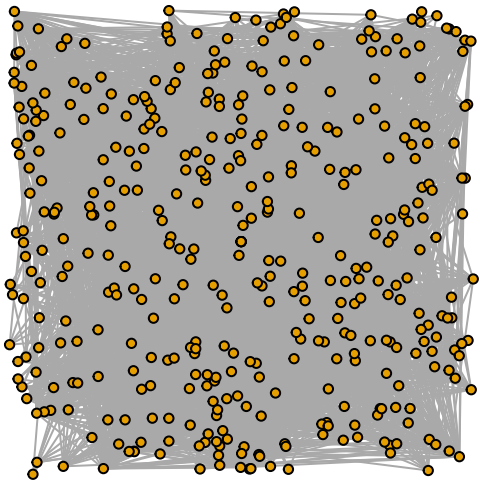
```
## Source: local data frame [7,913 x 3]
##
##         a      b     score
##     (chr)  (chr)     (dbl)
## 1     100    637 0.7333333
## 2  100015 100022 0.6000000
## 3  100029  99562 0.4736842
```

```
## 4  100041  82320 0.3333333
## 5   10005  11749 0.9230769
## 6  100065  97494 1.0000000
## 7  100109 114224 0.6976744
## 8  100109  96451 0.6521739
## 9  100109  97906 0.5172414
## 10 100109  99888 0.6250000
## ..    ...    ...       ...
```

Des paires choisies donnent des valeurs de seuil généralement supérieures au seuil du modèle.

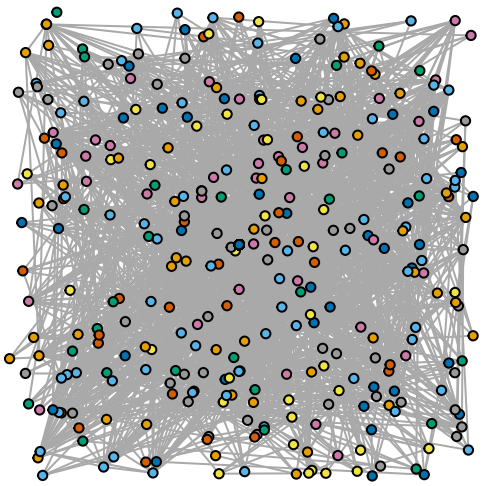
On visualise les relations (paires de candidats) par le graphe. On filtre les noeuds importantes dont le degré est supérieur à 4.

```
gr  <- graph( edges=a+1,  directed=F )
gr2 <- delete.vertices(gr,which(degree(gr)<4))
plot(gr2, vertex.size=4, vertex.label=NA, layout= layout.lgl)
```



Cherher des composants (clusters) dans ce graphe et les colorer.

```
clusters <- clusters(gr2)
plot(gr2, vertex.color = clusters$membership, vertex.size=4, vertex.label=NA,
     layout= layout.lgl)
```



Il y a 884 composants connexes qui sont détectés dans ce graph.

Pour chaque cluster, les 15 mots les plus fréquents sont affichés. Les résultats pour les 10 premiers clusters sont présentés.

```
for (i in 1: 10) {
  print(paste(" Pour le cluster ", i, "les mots specifiques sont: " ))
  doc            <- which(clusters$membership== i)
  corpus_cluster <- Venronp[doc]

  mots           <- findFreqTerms(TermDocumentMatrix(corpus_cluster) ,
                                  lowfreq = clusters$csize[i]/2, highfreq = Inf)
  print(na.omit(mots[1:15]))
  print("=================================================================")}
```

```
## [1] " Pour le cluster  1 les mots specifiques sont: "
##  [1] "aep"     "cnr"     "energi"  "est"     "file"    "financi" "gas"
##  [8] "imbal"   "market"  "mcv"     "nation"  "servic"  "tenaska" "trader"
## [15] "xls"
## [1] "================================================================="
## [1] " Pour le cluster  2 les mots specifiques sont: "
##  [1] "agreement" "award"     "basi"      "bid"       "capac"
##  [6] "commiss"   "day"       "discount"  "fts"       "person"
## [11] "pipelin"   "post"      "thing"     "time"
## attr(,"na.action")
## [1] 15
## attr(,"class")
## [1] "omit"
## [1] "================================================================="
## [1] " Pour le cluster  3 les mots specifiques sont: "
##  [1] "attach"     "california" "compani"    "electr"     "energi"
##  [6] "gas"        "group"      "kay"        "monday"     "money"
## [11] "nation"     "origin"     "pacif"      "prepay"     "referenc"
## [1] "================================================================="
## [1] " Pour le cluster  4 les mots specifiques sont: "
##  [1] "barbara" "charg"   "chris"   "corp"    "demand"  "germani" "intend"
##  [8] "kim"     "recipi"  "report"  "sabra"
## attr(,"na.action")
## [1] 12 13 14 15
## attr(,"class")
## [1] "omit"
## [1] "================================================================="
## [1] " Pour le cluster  5 les mots specifiques sont: "
##  [1] "boyt"      "deal"      "eric"      "gas"       "letter"
##  [6] "mcmichael" "payment"   "sell"      "tco"       "time"
## attr(,"na.action")
## [1] 11 12 13 14 15
## attr(,"class")
## [1] "omit"
## [1] "================================================================="
## [1] " Pour le cluster  6 les mots specifiques sont: "
##  [1] "agreement" "calc"      "cgas"      "compress"  "confirm"
##  [6] "corp"      "deal"      "exhibit"   "file"      "fuel"
## [11] "intend"    "nabisco"   "note"      "recipi"    "servic"
## [1] "================================================================="
## [1] " Pour le cluster  7 les mots specifiques sont: "
```

```
## [1] "call"    "deal"    "dth"     "kay"     "lee"     "prepay"  "reliant"
## [8] "termin"
## attr(,"na.action")
## [1]  9 10 11 12 13 14 15
## attr(,"class")
## [1] "omit"
## [1] "============================================================="
## [1] " Pour le cluster  8 les mots specifiques sont: "
##  [1] "busi"    "chris"   "compani" "converg" "counsel" "deal"    "dynegi"
##  [8] "energi"  "enov"    "fax"     "gas"     "handl"   "houston" "includ"
## [15] "invoic"
## [1] "============================================================="
## [1] " Pour le cluster  9 les mots specifiques sont: "
##  [1] "agre"       "capac"      "cent"       "chang"      "comment"
##  [6] "edison"     "place"      "rate"       "settlement" "time"
## attr(,"na.action")
## [1] 11 12 13 14 15
## attr(,"class")
## [1] "omit"
## [1] "============================================================="
## [1] " Pour le cluster  10 les mots specifiques sont: "
##  [1] "babi"    "call"    "contact" "corp"    "deal"    "ena"     "file"
##  [8] "gas"     "intend"  "jtowns"  "kay"     "list"    "payment" "petit"
## [15] "post"
## [1] "============================================================="
```

2. Latent Dirichlet Allocation

L'allocation de Dirichlet Latent appartient aux méthodes de *topic models*. Ces modèles permettent de décrire un document en fonctions de *topics* (thèmes) abstraits présent dans le corpus. Ainsi, lorsqu'un topic est dominant dans un document donné, on s'attend à ce que ce-dernier comporte beaucoup de mots appartenant à ce topic. Plus précisement dans un modèle LDA, les observations (le documents du corpus) sont expliquées par des groupes cachés (latents). Chaque document est considéré comme un mélange de thèmes générant alors chacun des mots du document. Le package `topicmodels` est utilisé pour réaliser le LDA.

Formalisation

Un *mot* est définit comme un élement du vocabulaire indexé $\mathcal{A} := \{1, .., V\}$. Un mot est définie comme une variable unitaire dont une unique composante vaut 1 et donc les autres composantes valent zéros. Des indices sont utilisées pour désigner les composantes d'un mot. Ainsi, le v-ème mot du dictionnaire est représenté par un vecteur de taille V et tel que $w^v = 1$ et $w^u = 0, \forall u \neq v$.

Un *document* est une N-uplet de mots dénoté $\boldsymbol{w} = (w_1, w_2, ..., w_N)$ où w_n est le n-ème mot de la séquence.

Un *corpus* est un ensemble de M documents dénoté: $\boldsymbol{D} = \{\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_D\}$

La variable $z_{d,n}$ représente le topic choisi pour le mot $w_{d,n}$ (le n-ème mot du document d) . θ_d est le paramètre gouvernant la distribution des topics dans le document d. α et η les *priors* sur les paramètres β et θ, β_k est la distribution du topic k.

Présentation du modèle

Les Processus Générateur des Données pour un document $\mathbf{d}$ du corpus $\boldsymbol{D}$ est le suivant:

1. Choisir $\beta \sim \text{Dir}(\eta)$
2. Choisir $\theta_d \sim \text{Dir}(\alpha)$, qui représente la proportion des K différents thèmes du document d.
3. Pour chacun des mot $w_{d,n}, n = 1, ..., N$:
   (a) Choisir un topic d'après les proportions tirées $z_{d,n} \sim \mathcal{M}(\theta_d)$
   (b) Choisir un mot $w_{d,n}$ a partir de $p(w_n|z_n, \beta)$, une probabilité d'une loi multinomiale conditionné par le topic z_n. Cela est équivalent à tirer dans $\mathcal{M}(\beta_k)$ avec $k = z_{d,n}$, où chacun des thèmes β_k est distribution multivariée de taille V sur les mots du vocabulaire $\mathcal{A}$.

Quelques remarques:

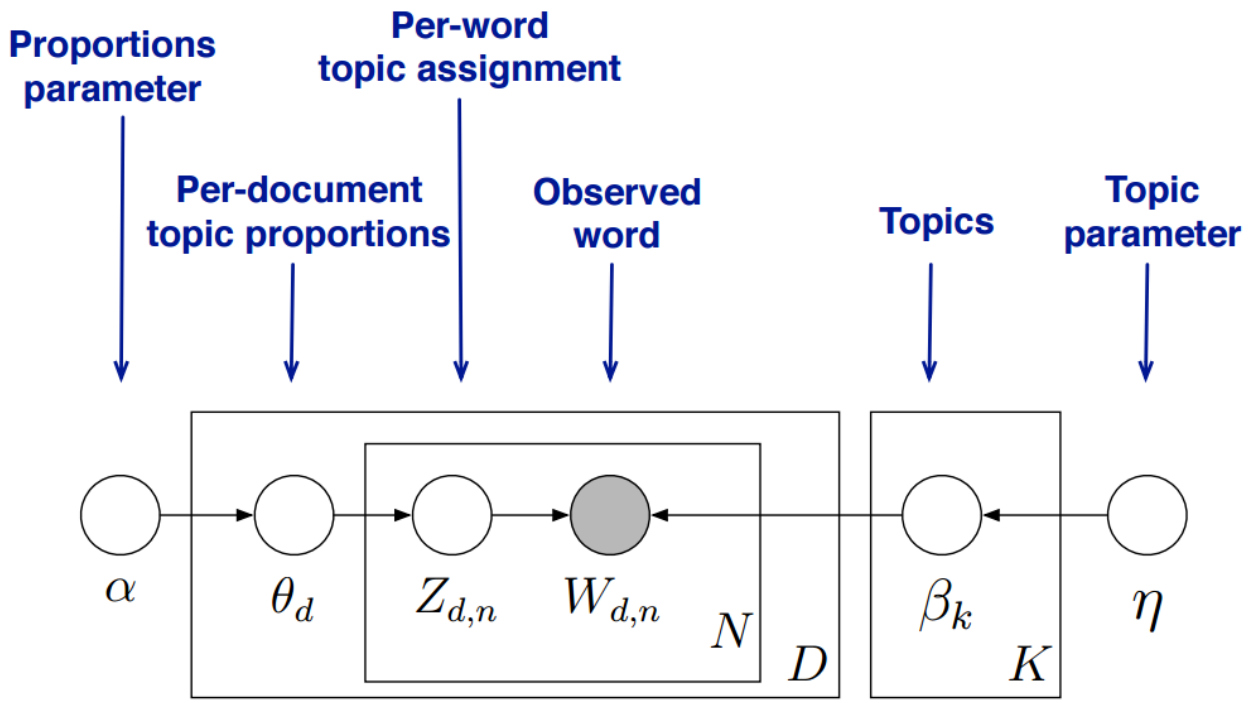
La dimensionnalité de la loi de Dirichlet, K est fixé. Autrement dit, le nombre de thèmes est fixé à l'avance.

Les probabilités des mots $w_{d,n}$ forment une matrice $k * V$ dont les composantes $\beta_{i,j} = p(w^j = 1|z_i = 1)$ sont des quantités fixé devant être estimées.

La distribution de Dirichlet a pour avantage d'être conjuguée avec la loi multinomiale, ce qui simplifie les calculs.

α et β sont des paramètres au niveau du corpus, supposé tiré une fois. Les variables θ_d au niveau des documents, tirées une fois pour chaque document. Enfin les variables $w_{d,n}, z_{d,n}$

Résumé du modèle graphique



Implémentation sur R

Chargement du corpus

```
enron    <- VCorpus(DirSource(enronpath, encoding= "UTF-8"),
                    readerControl=list(language= "en"))
```

Création de la Document Term Matrix et retrait des mots les moins fréquents.

```
dtm <- DocumentTermMatrix(enron)
d      <- removeSparseTerms(dtm,0.997) #inutile pour les petits corpus
d      <- d[as.matrix(rollup(d, 2, na.rm=T, FUN = sum))>0,]
```

Algorithme LDA

```
d         <- d[as.matrix(rollup(d, 2, na.rm=T, FUN = sum))>0,]
lda.model <- LDA(d, 6)
```

Les posteriors

```
posterior(lda.model)$terms[1:6,sample(2559,20)]
posterior(lda.model)$topics[sample(37068,20),1:6]
```

Résultats pour K=6

Nombre de documents: 37068, nombre de mots: 2558, sparsité : 0.997

Les 15 mots les plus probables dans les topics

Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6
power	message	deal	energy	attached	meeting
gas	intended	busy	company	agreement	vince
california	good	deals	trading	doc	time
state	don	hou	gas	corp	call
contract	recipient	day	power	kay	jeff
market	time	chris	market	fax	group
energy	bit	book	risk	hou	john
capacity	confidential	access	financial	america	week
rate	night	month	business	file	work
ferc	content	questions	price	legal	team
contracts	love	check	management	north	hou
price	work	appointment	year	ena	office
customers	day	number	products	sara	business
project	version	kate	companies	draft	conference
utilities	transfer	call	credit	phone	mike
commission	great	change	trade	copy	houston
order	hope	price	markets	letter	mark
demand	folder	desk	net	comments	forward
costs	weekend	file	news	houston	kaminski
time	encoding	list	stock	review	david

Les distributions posteriors des topics sur une sélection de mots

Les distributions posteriors des topics sur une sélection de documents

	short	complex	essentially	item	flow	cases	bear	heading
1	0.0014225	0.0001546	0.0002282	0.0003472	0.0007933	0.0006015	0.0001558	0.0000243
2	0.0006656	0.0000139	0.0000018	0.0000729	0.0000000	0.0002087	0.0001800	0.0002957
3	0.0006464	0.0001197	0.0000000	0.0005280	0.0013436	0.0000000	0.0000104	0.0000387
4	0.0021879	0.0003377	0.0001487	0.0000312	0.0004821	0.0002627	0.0001698	0.0002191
5	0.0001768	0.0000000	0.0000000	0.0003247	0.0000160	0.0000015	0.0004505	0.0000157
6	0.0006312	0.0000401	0.0000697	0.0000888	0.0000993	0.0000784	0.0000000	0.0000782

	1	2	3	4	5	6
61874.txt	0.4701	0.0315	0.0025	0.2128	0.0025	0.2806
108974.txt	0.0504	0.0504	0.7483	0.0503	0.0503	0.0503
15746.txt	0.6008	0.0424	0.0026	0.0026	0.0026	0.3489
49323.txt	0.0774	0.6130	0.0774	0.0773	0.0775	0.0774
11274.txt	0.0086	0.0801	0.0855	0.0086	0.8087	0.0086
38320.txt	0.0297	0.0297	0.0297	0.0299	0.3871	0.4939
86218.txt	0.0184	0.0184	0.0184	0.0183	0.0184	0.9081
75170.txt	0.0032	0.0032	0.3098	0.0031	0.0031	0.6776
14943.txt	0.0012	0.7070	0.2883	0.0012	0.0012	0.0012
18837.txt	0.0038	0.0038	0.0038	0.4130	0.0038	0.5717

Annexe 1: code Python

```
docs = []
from os import listdir, chdir, getcwd
import re, codecs, stop_words
import numpy as np
path_input = "/home/kintzler/R/projet_boucheron/maildir/"
path_output = "/home/kintzler/R/projet_boucheron/DATA/"

stopwords = stop_words.get_stop_words('en')
stopwords.extend("asap http www fyi hotmail email e-mail sent forwarded msn asked find send sent ect ma
stopwords.extend(open("stopwords.txt","r").read().split())
stopwords = sorted(list(set(stopwords))) #supprime les doublons

# Here's my attempt at coming up with regular expressions to filter out
# parts of the enron emails that I deem as useless.
email_pat = re.compile(".+@.+")
to_pat = re.compile("To:.+\n")
cc_pat = re.compile("cc:.+\n")
subject_pat = re.compile("Subject:.+\n")
from_pat = re.compile("From:.+\n")
sent_pat = re.compile("Sent:.+\n")
received_pat = re.compile("Received:.+\n")
ctype_pat = re.compile("Content-Type:.+\n")
reply_pat = re.compile("Reply- Organization:.+\n")
date_pat = re.compile("Date:.+\n")
xmail_pat = re.compile("X-Mailer:.+\n")
mimver_pat = re.compile("MIME-Version:.+\n")
fwd_pat = re.compile("Forwarded:.+\n")
```

```
chdir(path_input)
names = [d for d in listdir(".") if "." not in d]
for name in names:
    print(name)
    chdir(path_input+"%s" % name) #"%s"=string
    subfolders = listdir('.')
    sent_dirs = [n for n, sf in enumerate(subfolders) if "sent" in sf]
    sent_dirs_words = [subfolders[i] for i in sent_dirs]
    for d in sent_dirs_words:
        chdir(path_input+'%s/%s' % (name,d))
        file_list = listdir('.')
        docs.append([" ".join(codecs.open(f, 'r','latin-1').readlines()) for f in file_list if "." in f

docs_final = []
for subfolder in docs:
    for email in subfolder:
        if ".nsf" in email:
            etype = ".nsf"
        elif ".pst" in email:
            etype = ".pst"
        email_new = email[email.find(etype)+4:]

        email_new = to_pat.sub('', email_new)
        email_new = cc_pat.sub('', email_new)
        email_new = subject_pat.sub('', email_new)
        email_new = from_pat.sub('', email_new)
        email_new = sent_pat.sub('', email_new)
        email_new = email_pat.sub('', email_new)
        if "-----Original Message-----" in email_new:
            email_new = email_new.replace("-----Original Message-----","")
        email_new = ctype_pat.sub('', email_new)
        email_new = reply_pat.sub('', email_new)
        email_new = date_pat.sub('', email_new)
        email_new = xmail_pat.sub('', email_new)
        email_new = mimver_pat.sub('', email_new)
        email_new = fwd_pat.sub('', email_new)
        email_new = re.sub("[^a-zA-Z]", " ", email_new)
        email_new = ' '.join([word.lower() for word in email_new.split() if word.lower() not in stopwor
        email_new = ' '.join([word for word in email_new.split() if len(word)>2 and len(word)<30])
        email_new = re.sub(' +',' ',email_new)
        docs_final.append(email_new)

# Here I proceed to dump each and every email into about 126 thousand separate
# txt files in a newly created 'data' directory.  This gets it ready for entry into a Corpus using the
# package from R.



for n, doc in enumerate(docs_final):
    if np.random.rand()<0.3:
        outfile = open(path_output+"%s.txt" % n ,'w')
        outfile.write(doc)
        outfile.write('\n')
        outfile.close
```

```
print("Fin du script")
```