

**Cours** : Text et Web Mining (TMW). **Partie 1.**

**Niveau** : Master 1 TIC

**Enseignant** : Lyazid.SABRI

## Préambule

Le Texte Mining (TM) est un domaine de l'intelligence artificielle. Il combine de nombreux domaines, tels que les statistiques, la linguistique et le traitement automatique du langage naturel (TAL). Le TM est aujourd'hui beaucoup utilisé pour la recherche d'informations, extraction de connaissances à partir d'un corpus de texte, exploration du web, etc. Il est également utilisé pour l'analyse de données (données médicales, tendances, sentiments, opinions, marketing, etc.) et l'annotation de documents. Cette dernière, est primordiale pour une analyse d'un texte. L'extraction d'information quant à elle, est un champ de recherche ouvert, puisqu'il s'agit entre autres de trouver des relations sémantiques et de causalité dans un texte rédigé en langage naturel (i.e. documents non structurés).

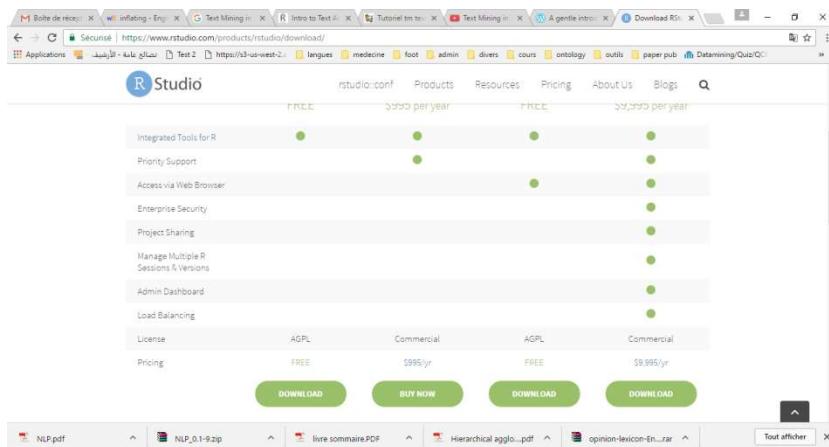
Pour mettre en œuvre le processus de classification et catégorisation, le TM distingue entre l'apprentissage supervisé et non supervisé. La notion d'apprentissage supervisée cherche à prédire la classe d'une source (document, observation, etc.) à partir d'un échantillon de classes prédefinies. L'apprentissage non supervisé, permet quant à lui de déterminer à quelle classe un document, un thème, etc. puisse être affecté (i.e. il n'existe pas de classes prédefinies). Il existe de nombreuses méthodes pour le traitement de textes tant pour la classification que pour la catégorisation, certaines sont fondées sur des méthodes statistiques, probabilistes, linguistique ou fondée sur des approches d'apprentissage automatiques (ML : Machine learning, en Anglais).

Dans ce document, vous allez utiliser le langage R pour appréhender les principes de bases de fouilles de textes. Pour faire suite à ce document, je présenterai d'autres outils intéressant à connaître et d'autres algorithmes dédiés aux fouilles de textes, et en particulier les approches fondées sur les ontologies et le deep learning.

Des exemples simples seront utilisés et étudiés tout au long de ce document. Toutefois, pour une meilleure compréhension, le lecteur est invité à lire de manière séquentielle et devrait absolument reproduire les tests et faire les exercices dans l'ordre.

## Installation de RSTUDIO & le langage R.

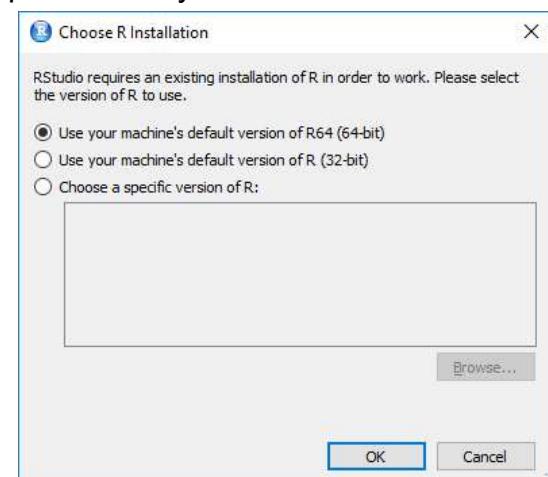
RStudio est un ensemble d'outils pour faciliter l'utilisation du langage R. vous devrez donc aller sur le site : <https://www.rstudio.com/products/rstudio/download/>, et vous rendre à la page Download (figure ci-dessous), et téléchargez l'outil R Studio 32 ou 64 bits selon votre système.



Après avoir téléchargé R Studio.exe installez-le dans un répertoire à votre convenance. A chaque étape de l'installation, faire un clic sur suivant, et surtout ne changer rien aux valeurs par défauts lors de votre installation. Enfin, ne pas lancer RStudio avant de télécharger et installer le langage R pour votre système.

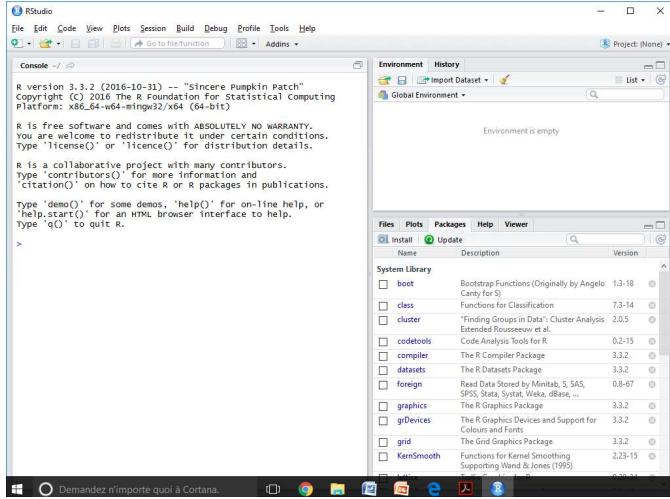
Aller sur le site suivant, <https://cran.r-project.org/bin/windows/base/>, télécharger le langage R et lancer l'installation. Une fois l'installation terminée, lancez R Studio.

Une fenêtre ressemblant à celle en face devrait apparaître.



Sélectionnez le processeur de votre station et valider votre choix.

Une interface graphique comme celle illustrée par la figure suivante devrait apparaître.



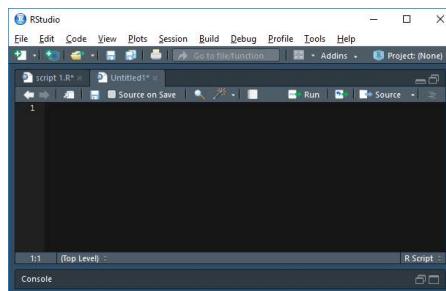
Vous pouvez saisir dans la partie console une première commande pour s'en assurer que tout va bien. Par exemple, vous pouvez vérifier le répertoire de travail via la commande suivante `getwd()` (i.e. get working directory).

## Création de fichiers scripts avec R.

La meilleure façon d'utiliser le langage R est de créer un script pour chaque manipulation. Ce script peut être sauvegardé et être réutilisé plus tard. Ne pas oublier d'insérer des commentaires en commençant chaque ligne de code par le symbole `#`.

Afin de créer un fichier script, il suffit d'aller dans le menu *File ->new file-> R script*.

Une zone d'édition s'affiche comme illustrer par la figure suivante.



Enregistrer votre fichier sous un nom reflétant le contenu de votre fichier.

Avant de commencer et à chaque fois que c'est nécessaire, faire un *Ctrl+L* pour supprimer le contenu de la console. Pour visualiser cet onglet (console), il suffit d'utiliser la sourie pour en avoir deux parties dans l'interface RStudio.

Une des premières commandes que vous devez tester est : *getwd()*. Une fois la commande saisie dans la zone d'édition, il suffit de faire un clic sur *Run*. La commande sera automatiquement exécutée. Le résultat sera affiché sur la console. Vous pouvez également sélectionner la commande ou pointer sur la fin de ligne ou le début de ligne et faire *Ctrl+Retour\_Chariot* au lieu de faire un clic sur *Run*.

Nous allons devoir installer quelques packages (librairies) utiles au bon déroulement de nos tests. Pour ce faire deux méthodes se présentent à vous :

1. Saisir la commande *install.packages(" SAISIR LE NOM DU PACKAGE")* ou plusieurs packages comme suit : *install.packages("PACKAGE 1"," PACKAGE 1", ...)*
2. Utiliser l'assistant *RStudio* : *Tools-> install Packages...*

## Utilisation du package TM avec R.

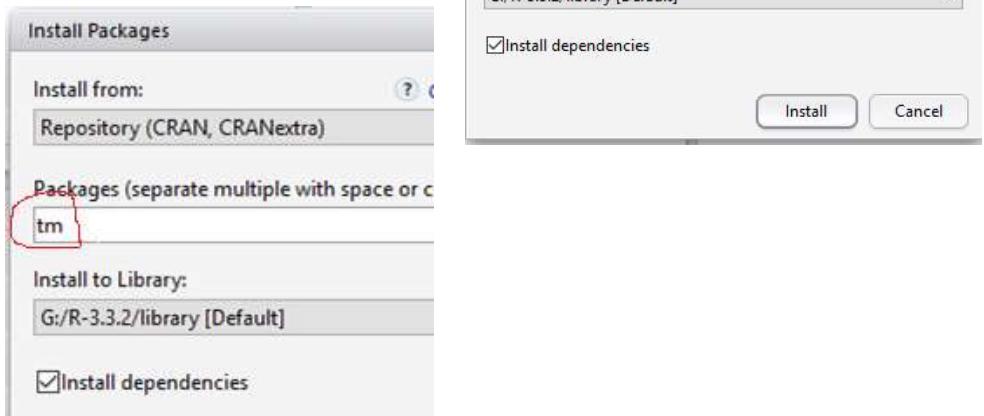
Vérifier dans un premier temps que vous avez accès à Internet. Si vous utiliser le second moyen d'installation, alors il faut refaire ces étapes autant de fois que vous avez de librairies à installer. L'avantage de cette méthode est qu'elle permet de vérifier instantanément que la librairie est installée. Quant à la première méthode, bien qu'elle permette de saisir les noms de toutes les librairies que l'on souhaite installées, cependant, si lors d'un échec d'installation (surtout si on ne se trouve pas devant son pc), il faudra remonter la console pour lire et vérifier.

Personnellement quand il s'agit de moins de deux packages, j'utilise la première méthode, mais quand il s'agit d'un nombre important de librairies, j'utilise la seconde méthode. Une des librairies les plus utilisées pour la fouille de textes est *tm* (text mining). Pour plus de détails sur cette librairie, vous pouvez vous rendre à cette page : <https://cran.r-project.org/web/packages/tm/tm.pdf>

Aller à *Tools-> install Packages...* La boite de dialogue comme celle illustrée par la figure suivante devrait apparaître.

Saisir *tm* dans la zone Packages (....) comme suit :

Cocher la case *Install dependencies*, si ce n'est pas le cas. Ensuite, faire un clic sur *Install*.



Vérifier dans la console de R studio que l'installation s'est bien déroulée. Pour ce faire, il suffit de charger la librairie comme suit : library (tm). Sur la console, il faut vérifier que le chargement s'est bien déroulé et qu'aucune exception ou erreur n'est signalée.

**Remarque :** il est possible qu'une librairie dépende d'une ou plusieurs autres librairies. Cette information est signalée par R sur la console. Ainsi, il faut procéder à l'installation de cette librairie requise (voir plusieurs) pour pouvoir installer la librairie que vous avez voulu installer au départ.

Vous devez également installer les librairies suivantes. D'autres seront indiquées au fur à mesure que vous avancez dans la lecture de ce document. Vous pouvez lire rapidement la description et l'utilisation de chaque librairie en allant sur les liens indiqués.

- *quanteda* : voir le site <https://quanteda.io/articles/pkgdown/quickstart.html>
- *devtools*
- `install.packages("spacyr")`

- `install.packages("readtext")`
- `devtools::install_github("quanteda/quanteda.corpora")`
- `qdap`
- *SnowballC* pour effectuer le stemming. <https://cran.r-project.org/web/packages/SnowballC/SnowballC.pdf>
- *Textstem* : <https://cran.r-project.org/web/packages/textstem/textstem.pdf>
- *ggplot2* : Pour rendu graphique des données <https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf>
- *wordcloud* : ce dernier réalisera un effet visuel présentant en premier plan des mots les plus pertinents dans un corpus. <https://cran.r-project.org/web/packages/wordcloud/wordcloud.pdf>
- *Pdftools* : pour la gestion des fichiers pdfs. <https://cran.r-project.org/web/packages/pdftools/pdftools.pdf>
- `install.packages("tokenizers")` : Joue un rôle primordial dans l'analyse lexicale. Bien que ce terme « token » peut signifier différents choses selon le domaine, cependant dans le cadre de fouilles de textes, il permet de découper un texte en mots, symbole, ou phrases appelés token.  
<https://cran.r-project.org/web/packages/tokenizers/tokenizers.pdf>
- `install.packages("tidyverse")` : fournit des outils pour la lecture et la gestion des données. <https://cran.r-project.org/web/packages/tidyverse/tidyverse.pdf>

## **Installation de TreeTagger.**

TreeTagger est un outil développé (1994-1995) par *Helmut Schmid* à l'Institut de linguistique computationnelle de l'Université de Stuttgart pour annoter du texte avec des informations sur la partie du discours. TreeTagger a été utilisé avec succès pour étiqueter de nombreuses langues telles que l'allemand, l'anglais, le français, l'italien, etc. Il peut être adapté à d'autres langues si un lexique et un corpus de formation balisés manuellement sont disponibles.

### **a. Quelques commentaires sur TreeTagger.**

En raison de sa grande fiabilité, le corpus utilisé pour la phase d'apprentissage de TreeTagger est le Penn Treebank [1]. Il s'agit d'un corpus d'anglais natif composé de 4,5 millions de mots en anglais américain et correspondant aux articles parus dans le Wall Street Journal. Ce corpus a fait l'objet d'une annotation syntaxique décrivant un système d'arbre de dépendances syntaxiques, et d'une annotation PoS (Part of Speech qui signifie analyse de discours). Le jeu d'étiquettes PoS, qui comporte 36 étiquettes PoS et 12 pour la ponctuation et les symboles monétaires, été appliqué automatiquement sur le corpus avant qu'une procédure de correction manuelle ne soit mise en œuvre pour permettre à des annotateurs humains de modifier les étiquettes erronées [2].

### **b. De quoi s'agit –il exactement**

Cet outil a pour objectif l'annotation d'un texte avec des informations sur la partie de discours (PoS). Il s'agit ici d'indiquer les différentes catégories d'un mot telles que nom, verbe, etc. Cet étiquetage morphosyntaxique consiste à associer une étiquette morphosyntaxique à chaque mot. Ce procédé s'appuie sur la segmentation en mots et en phrases effectuée préalablement. Le mécanisme de lemmatisation (voir plus bas dans ce document) s'appuie à son tour sur l'analyse morphosyntaxique, et est effectué si un mot du texte peut être annoté grammaticalement, dans le cas contraire, aucune information n'est associée à ce mot. Pour plus d'informations aller à la page suivante : <https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/Penn-Treebank-Tagset.pdf>

En général, PoS utilisé dans Pen Treebank par TreeTagger associe (pour la langue anglaise) à chaque mot les annotations suivantes :

Tag	Description	Tag	Description
CC	Coordinating conjunction	NNPS	Proper noun, plural
CD	Cardinal number	PDT	Predeterminer
DT	Determiner	POS	Possessive ending
EX	Existential <i>there</i>	PRP	Personal pronoun
FW	Foreign word	PRP\$	Possessive pronoun
IN	Preposition or subordinating conjunction	RB	Adverb
JJ	Adjective	RBR	Adverb, comparative
JJR	Adjective, comparative	RBS	Adverb, superlative
JJS	Adjective, superlative	RP	Particle
LS	List item marker	SYM	Symbol
MD	Modal	TO	<i>to</i>
NN	Noun, singular or mass	UH	Interjection
NNS	Noun, plural	VB	Verb, base form
NNP	Proper noun, singular	VBD	Verb, past tense
VBP	Verb, non-3rd person singular present	VBG	Verb, gerund or present participle
VBZ	Verb, 3rd person singular present	VBN	Verb, past participle
WDT	Wh-determiner	WP	Wh-pronoun
WRB	Wh-adverb	WP\$	Possessive wh-pronoun

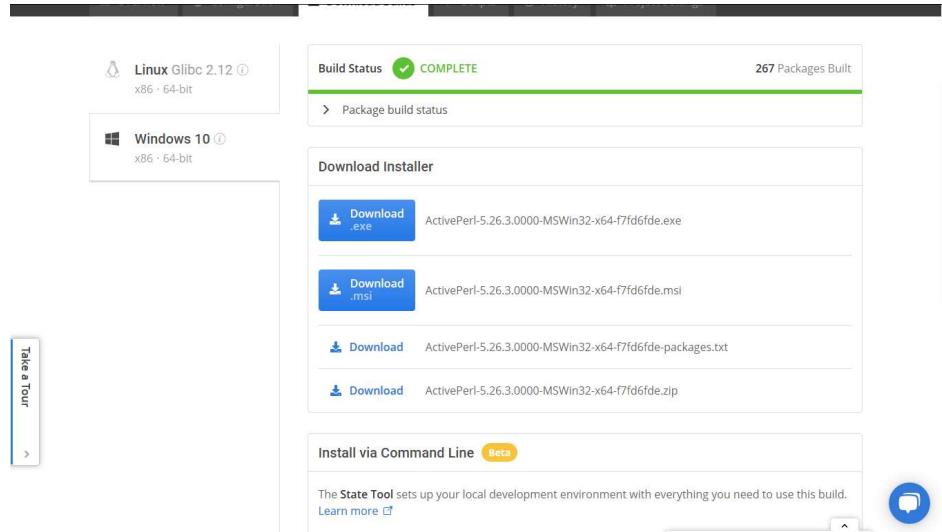
Aller à la page : <https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/> pour télécharger TreeTagger; pour celles ou ceux qui utilisent Linux peuvent suivre les instructions indiquées sur la page web.

Dans ce document j'utilise la version dédiée à Windows 64. Donc au bas de la page, faire un clic sur are available [for Windows64](#) and [for Windows32](#), afin de télécharger la version qui vous convienne.

Dans la section Parameter **files de la page, télécharger** le fichier de langue à utiliser. (Français et Anglais). Pour l'Anglais, récupérer PENN Treebank.

Aller sur le site <http://www.activestate.com/activeperl/> pour télécharge la version de Perl interperter (nécessaire pour TreeTagger). Vous serrez rediriger pour créer un compte, ensuite être capable de télécharge la version adéquate.

Attention, il y a aussi une version pour Linux et une autre pour Windows. Si vous faites le choix pour Windows, il se peut que vous serriez amener à accepter les règles d'activation, mais attention, tout est gratuit.



Une fois Perl téléchargé, et lors de l'installation, laissez-vous guider et laisser le répertoire par défaut.

Maintenant, Dé-zipper le fichier TreeTagger, et mettre ce fichier dans la racine du C. (vous pouvez le mettre dans un autre répertoire de votre PC, mais, vous serez obligé de faire d'autres configurations non expliquées dans ce document).

Ajouter la valeur C:\TreeTagger\bin dans le PATH de la variable d'environnement de votre système.

Dé-zipper les fichiers de langues, et copier les fichiers dont l'extension est (.par) et collez-les dans le TreeTagger/lib. Ce dernier répertoire, contient les fichiers de paramètres utilisés pour la catégorisation grammaticale des mots. Quant au répertoire cmd, des programmes Perl avant la phase d'étiquetage pour segmenter un texte et un autre programme pour la conversion des résultats en format XML.

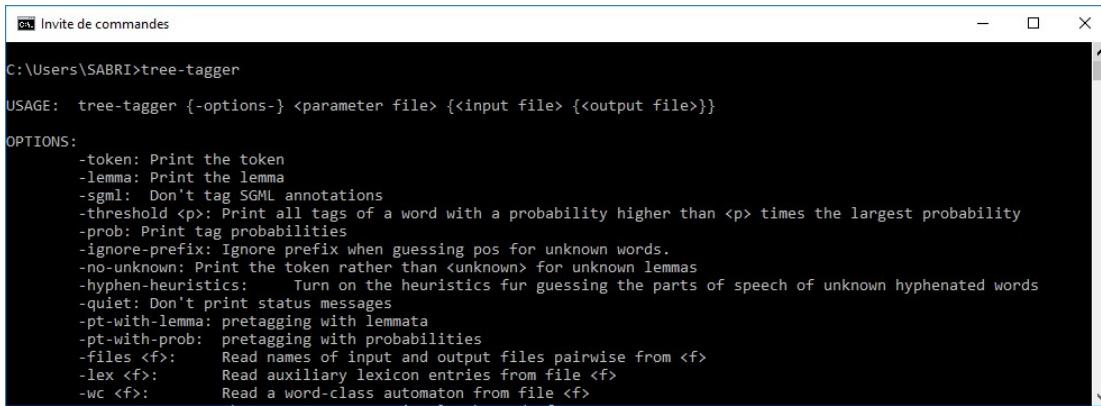


TreeTagger est fourni sans interface graphique, par conséquent, pour celles ou ceux qui souhaitent avoir une interface graphique, peuvent suivre les étapes indiquées dans le lien :

<http://www.smo.uhi.ac.uk/~oduibhin/oideasra/interfaces/winttinterface.htm>.

### **Exemple d'exécution**

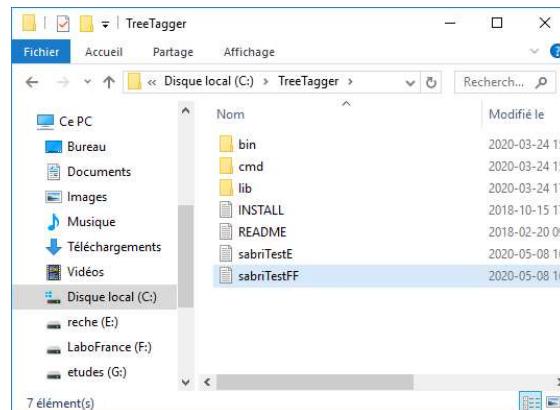
Ouvrir une CMD (invité de commande), et exécuter la commande tree-tagger. Une fenêtre comme celle présentée par la figure suivante montre les options pouvant être utilisées.



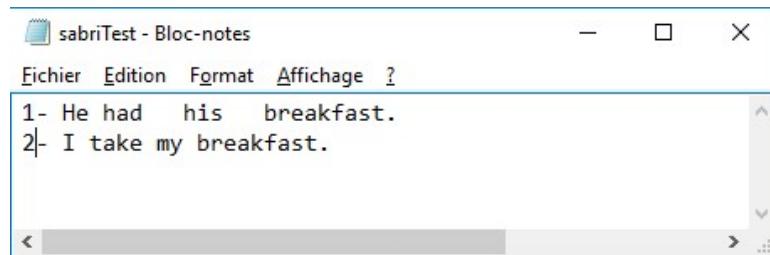
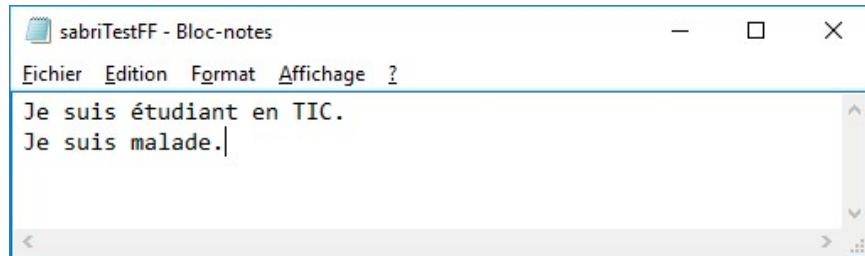
```
C:\Users\SABRI>tree-tagger
USAGE: tree-tagger {-options-} <parameter file> {<input file>} {<output file>}

OPTIONS:
  -token: Print the token
  -lemma: Print the lemma
  -sgml: Don't tag SGML annotations
  -threshold <p>: Print all tags of a word with a probability higher than <p> times the largest probability
  -prob: Print tag probabilities
  -ignore-prefix: Ignore prefix when guessing pos for unknown words.
  -no-unknown: Print the token rather than <unknown> for unknown lemmas
  -hyphen-heuristics: Turn on the heuristics for guessing the parts of speech of unknown hyphenated words
  -quiet: Don't print status messages
  -pt-with-lemma: pretagging with lemmata
  -pt-with-prob: pretagging with probabilities
  -files <f>: Read names of input and output files pairwise from <f>
  -lex <f>: Read auxiliary lexicon entries from file <f>
  -wc <f>: Read a word-class automaton from file <f>
```

Rédiger deux textes avec l'éditeur bloc note. Copier ces fichiers dans le répertoire racine de TreeTagger (pour faire vite c'est tout). Dans la figure suivante, vous devez lire le fichier de teste pour l'exemple. Dans mon cas, mes deux fichiers sont nommés sabriTest E (English) et sabriTest FF (Français)



Ce fichier nommé sabriTest contient deux phrases : (attention à l'accentuation), remarquer que j'ai mis des chiffres pour faciliter la lecture des résultats



Saisir les commandes suivantes et observer les résultats :

**Commande 1 :**

```
C:\TreeTagger>perl cmd/tokenize.pl sabriTestF.txt | bin\tree-tagger -token -lemma lib\francais.par
```

```
C:\Windows\System32\cmd.exe
C:\TreeTagger>perl cmd/tokenize.pl sabriTestFF.txt | bin\tree-tagger -token -lemma lib\francais.par
      reading parameters ...
      tagging ...
    Je  ADJ    <unknown>
suis  VER:pres    suivre|>-tre
étudiant  NOM    |@étudiant
en   PRP    en
TIC   NOM    tic
.    SENT   .
Je   PRO:PER je
suis  VER:pres    suivre|>-tre
malade  ADJ    malade
.    SENT   .
                           finished.

C:\TreeTagger>
```

**Question 1:** en lisant le résultat attentivement, qu'est ce que vous remarquez?

**Seconde commande :**

```
C:\TreeTagger>perl cmd/tokenize.pl sabriTestE.txt | bin\tree-tagger -token -lemma lib\anglais.par
```

```
C:\TreeTagger>perl cmd/tokenize.pl sabriTestE.txt | bin\tree-tagger -token -lemma lib\english.par
reading parameters ...
tagging ...
1- CD @card@
He PP he
had VHD have
his PP$ his
breakfast NN breakfast
.
2- CD @card@
I PP I
take VVP take
my PP$ my
breakfast NN breakfast
.
SENT .
finished.

C:\TreeTagger>
```

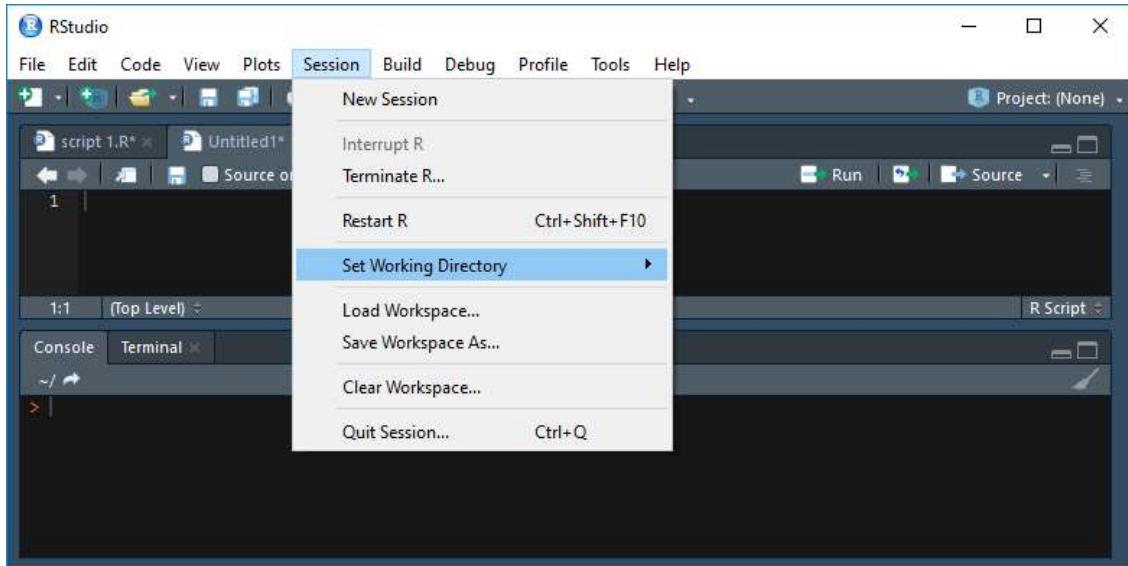
Comme réponse à la question ci-dessus: Pas de contexte : le mot « suis » signifie, je marche derrière un étudiant, et que cet étudiant est dans la spécialité TIC, ou, ce verbe « suis », signifie que je fais partie des étudiants, et j'apprends des cours de spécialité TIC.

Les outils installés jusqu'ça présent seront mis en œuvre dans le langage R. avant de commencer, nous avons besoins d'utiliser des documents non structurés écrits en langue Anglaise. Quant à la langue Française, nous allons utiliser d'autres outils.

Vu le nombre important de textes, je ne peux pas mettre les textes dans ce document, mais seront remis avec ce document. Par conséquent, vous devez télécharger les fichiers de travail dans un répertoire nommé docs2. Dans un premier temps commencer d'abord à lire ces textes pour se faire une idée de leur contenu. Ensuite, renseigner dans le R Studio le répertoire de travail en exécutant une des deux commandes suivantes:

1. setwd (le chemin de votre dossier) (méthode que je préconise)
2. Une autre méthode consiste à utiliser le menu session

**Remarque :** en utilisant une session, R crée des fichiers supplémentaires dans vote répertoire.  
**Remarque :** pour celle ou celui qui utilise la première méthode, alors sous Windows, le chemin de votre fichier doit utiliser un double séparateur \\. Par exemple : setwd("G:\\TM\\docs2"). A la fin vérifier le chemin avec getwd.



Avant de continuer, il est indispensable d'installer les packages comme expliquer au début de ce document.

Charger la librairie tm en exécutant la commande suivante :`library(tm)`

```
getwd()
library(tm)
```

The screenshot shows the RStudio console window. It has a toolbar with buttons for back, forward, search, and run. Below the toolbar, the console output is displayed. The first line 'getwd()' is shown in grey, indicating it's a command. The second line 'library(tm)' is shown in red, indicating it's a function call. The third line is a blank line starting with a '|', indicating the command is still executing or has just finished.

- `getStemLanguages()` : Cette commande permet de donner la liste des langues supportées.
- Charger la librairie SnowBallC
- Saisir dans le terminal de RStudio le mot *word*, une liste de proposition de commande devrait s'afficher.
- Sélectionner la commande `wordStem`.

Cette dernière commande, renvoie le stem de chaque mot du vecteur selon l'algorithme porter. Les éléments du vecteur sont convertis au codage UTF-8 avant que le stemming soit effectué, et les éléments retournés sont marqués comme tels lorsqu'ils contiennent des caractères non ASCII.

Il suffit de saisir entre les parenthèses de cette fonction une liste de mots ou utiliser la liste de mots dans une variable comme suit (par est un nom de variable que vous pouvez modifier) :

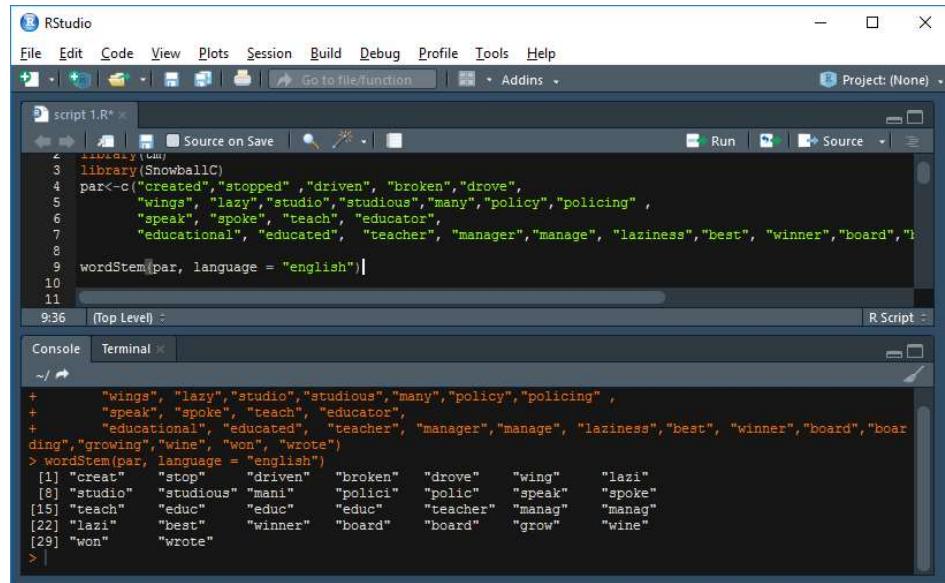
```
par<-c("created","stopped" , "driven", "broken","drove", "wings", "lazy","studio",
"studious", "many","policy","policing", "speak", "spoke", "teach", "educator",
"educational","educated", "teacher", "manager","manage", "laziness","best",
"winner","board", "boarding","growing", "wine", "won", "wrote")
```

Utilisé la commande suivante pour créer un vecteur de lemm :

```
lemmatize_words(par)
```

```
> lemmatize_words(par)
[1] "create"      "stop"       "drive"      "break"      "drive"      "wing"       "lazy"       "studio"
[9] "studious"    "many"       "policy"     "police"     "speak"      "speak"      "speak"
"teach"        "educator"
[17] "educational" "educate"    "teacher"    "manager"    "manager"    "manage"    "laziness"
"good"         "winner"
[25] "board"       "board"     "grow"       "wine"       "win"       "write"
```

Ensuite, exécuter la commande : wordStem(par, language = "english")



Cette commande peut être utilisée autrement.

```
wordStem(c("données","issues","maux", "mal", "mise","miser","mettre", "ligne",
"ligner","bas","base", "contrat","contracter", "sont", "peu","peut", "fiables") ,language =
"francais")
```

## Exercice

Essayer de mettre le résultat de la lemmatisation dans une variable, ensuite effectuer le stemming (wordStem) sur cette nouvelle variable. Que remarquez-vous ?

## Lemmatisation

Pour plus d'information sur la linguistique, lire le livre en ligne de cours de linguistique générale, éd. Bally et Sechehaye, 1971. (Publication originale 1916)

[https://fr.wikisource.org/wiki/Livre:Saussure\\_-\\_Cours\\_de\\_linguistique\\_g%C3%A9n%C3%A9rale,\\_%C3%A9d.\\_Bally\\_et\\_Sec...1971.djvu](https://fr.wikisource.org/wiki/Livre:Saussure_-_Cours_de_linguistique_g%C3%A9n%C3%A9rale,_%C3%A9d._Bally_et_Sec...)

Brièvement, un lemmatiseur tel que TreeTagger<sup>1</sup> associe à chaque lemme (nous parlons de token (mot)) la forme canonique. Il s'agit ici d'une analyse lexicale permettant de mettre par exemple des verbes à l'infinitif, la forme singulière, de déterminer le genre (masculin/féminin) pour les noms, etc. En outre, la lemmatisation est une technique pour diminuer ce que l'on appelle les flexions. Le mot flexion<sup>2</sup> désigne un changement morphologique dans la finale d'un mot (nom, pronom, participe, adjectif) selon la fonction qu'il occupe dans la phrase ou dans la proposition par l'adjonction d'un affixe, ou désinence<sup>3</sup> au radical ou au thème. Quand il s'agit d'une modification de l'élément de base du mot (changement de vocalisme, de timbre, d'accent, etc.), nous parlons également de flexion interne. La flexion externe, quant à elle se produit quand la base flexionnelle se voit adjoindre un indice morphologique sous la forme d'un suffixe [3]. À titre d'exemple, pour l'anglais, la base flexionnelle robot forme son pluriel en ajoutant l'indice s (flexion externe). Dans le cas d'une flexion interne, l'ouze qui devient l'ice (pou).

Enfin, il peut désigner un ensemble des formes fléchies d'un mot (nom, pronom ou verbe) variant selon le cas, le genre, le nombre et la personne, la flexion nominale, la flexion verbale. Par exemple, beaucoup de mots sont des unités complexes, où l'on

---

<sup>1</sup> Un des outils effectuant une lemmatisation de différentes langues

<sup>2</sup> Selon le centre national de ressources textuelles et lexicales

<sup>3</sup> Élément variable à la finale d'un mot, qui, ajouté au radical (ou au thème de flexion), sert à marquer chacune des formes verbales (dont l'ensemble constitue la conjugaison) ou nominales (dont l'ensemble constitue la déclinaison ou la flexion).

distingue aisément des sous-unités (suffixes, préfixes, radicaux); des dérivés comme désir-eux, malheur-eux se divisent en parties distinctes dont chacune a un sens et un rôle évidents. Donc, l'analyse morphosyntaxique des textes se traduit par différents niveaux d'analyse en traitement du langage naturel. En effet, l'interprétation de l'exemple "je suis un étudiant de Tic" est difficilement interprétable. C'est pourquoi, une interprétation n'est correcte que si elle est située dans le contexte du texte. Un autre exemple, serait dans le saint coran. Dans la sourate la *FUMÉE* (AD-DUKHAN) quand Dieu évoque le dialogue avec son prophète Moïse (Moussa). Dans ce dialogue Dieu ordonne Moïse de voyager de nuit, et comme suite dans cette sourate, Dieu s'adresse à son Prophète Mohammed (que le salut de Dieu soit sur lui). Ce dernier contexte est exprimé implicitement dans le versé 34 comme suit Au nom d'Allah le Tout Miséricordieux, le Très Miséricordieux :

*17 Et avant eux Nous avons déjà éprouvé le peuple de Pharaon, quand un noble Messager leur était venu, 18 [leur disant]: « Livrez-moi les serviteurs d'Allah ! Je suis pour vous un Messager digne de confiance. 19 Ne vous montrez pas hautains vis-à-vis d'Allah, car je vous apporte une preuve évidente. 20 Et je cherche protection auprès de mon Seigneur et votre Seigneur, pour que vous ne me lapidiez pas. 21 Si vous ne voulez pas croire en moi, éloignez-vous de moi. 22 Il invoqua alors son Seigneur: Ce sont des gens criminels. 23 Voyage de nuit avec Mes serviteurs; vous serez poursuivis. 24 Laisse la mer calme; [telle que tu l'as franchie] ce sont, des armées [vouées] à la noyade. 25 Que de jardins et de sources ils laissèrent [derrière eux] 26 que de champs et de superbes résidences, 27 que de délices au sein desquels ils se réjouissaient. 28 Il en fut ainsi et Nous fîmes qu'un autre peuple en hérita. 29 Ni le ciel ni la terre ne les pleurèrent et ils n'eurent aucun délai. 30 Et certes, Nous sauvâmes les enfants d'Israël du châtiment avilissant. 31 de Pharaon qui était hautain et outrancier. 32 À bon escient Nous les choisîmes parmi tous les peuples de l'univers, 33 et leur apportâmes des miracles de quoi les mettre manifestement à l'épreuve. 34 Ceux-là disent: 35 « Il n'y a pour nous qu'une mort, la première. Et nous ne serons pas ressuscités. 36 Faites donc revenir nos ancêtres, si vous êtes véridiques.*

Dans cette sourate : remarquons qu'un seul mot -Ceux-là (هؤلؤ)- fait référence aux Mecquois (polythéistes). La question, comment peut-on sous-entendre que Dieu s'adresse au prophète Mohammed (que le salut de Dieu soit sur lui)?

Ainsi, le traitement du langage naturel s'attaque à plusieurs niveaux d'analyse.

1. La morphologie : étudie la forme des mots ; il s'agit de traiter des diverses catégories de mots (verbes, noms, adjectifs, pronoms, etc.) et des différentes formes de la flexion (conjugaison, déclinaison)
2. La syntaxe structurale : s'intéresse à l'organisation des mots en vue de déterminer les unités qui composent un énoncé. La structuration des mots permet donc d'établir des relations hiérarchiques entre ces mots.
3. La sémantique : étudie le sens d'un mot
4. La pragmatique : le contexte est pris en compte lors de l'analyse du sens d'un mot.

Le quatrième point est primordial pour la bonne interprétation des énoncés. Pour comprendre ce que la pragmatique signifie, je vais reprendre les exemples du papier de Georges Kleiber paru en 1982 [4]. Dans son document, Kleiber expliquait que chez Peirce, la pragmatique n'est pas une composante de la linguistique, comme la phonologie, la syntaxe et la sémantique, mais constitue une théorie d'accueil unifiée pour toutes les branches de la linguistique. A l'intérieur d'une telle théorie maximaliste, tout fait linguistique devient finalement (dans le sens de « *est subordonné à* », ou « *est intégré dans* ») pragmatique. Quant à L. Apostel dont l'hypothèse sur l'intégration de la théorie du discours et de l'acte de communication dans une théorie générale de l'action devrait aboutir « à une grammaire et une sémantique (...) basée sur la théorie de l'action ». Par exemple, Dans la phrase : Pouvez-vous ouvrir la fenêtre? D'un point de vue sémantique, nous pouvons comprendre qu'une personne cherche à savoir si la personne à laquelle il demande d'ouvrir la fenêtre possède la capacité de l'ouvrir; toutefois, elle peut aussi signifier une demande d'ouvrir une fenêtre. Or, cette valeur pragmatique de requête n'est pas attachée à un emploi particulier de cette phrase, à une condition d'énonciation précise, individuelle, comme l'est par exemple la valeur de requête « *Ouvrez la fenêtre !* » que peut véhiculer la phrase "*il fait chaud ici*". Ainsi, bien que la lemmatisation diminue considérablement le nombre de forme de mots et leur occurrence pour l'analyse d'un corpus. Cependant, cette opération fait perdre de l'information résultant de la substitution d'un mot par son lemme, d'autant plus pour une analyse s'appuyant sur le contexte. Ce qui nous emmène à donner quelques définitions :

- mot<sup>4</sup> : un mot s'oppose à un morphème<sup>5</sup>. Un mot est un son ou groupe de sons articulés ou figurés graphiquement, constituant une unité porteuse de signification à laquelle est liée, dans une langue donnée, une représentation d'un être, d'un objet, d'un concept, etc. Les mots nous présentent des choses une petite image claire et usuelle, comme celles qu'on suspend aux murs des écoles, pour nous donner l'exemple de ce qu'est un établi, un mouton, un chapeau, choses conçues comme pareilles à toutes celles de même sorte.

Mais, en linguistique, un mot<sup>6</sup> est une unité significative indépendante, ne pouvant pas toujours être déterminée selon un critère de séparabilité fonctionnelle ni par un critère de délimitation intonative: Il faut chercher l'unité concrète ailleurs que dans le mot. Du reste beaucoup de mots sont des unités complexes, où l'on distingue aisément des sous-unités (suffixes, préfixes, radicaux). Inversement il y a des unités plus larges que les mots tel que, *porte-plume, s'il vous plaît*.

Pour Saussure, une langue est composée d'unités discrètes qui ne sont pas immédiatement perceptibles, mais qui doivent être identifiées par l'analyse, et qui définissent une combinatoire : ces unités, ce sont les signes, qui unissent chacun un signifié (concept) et un signifiant (image acoustique). Par exemple, les mots *mutton* et *sheep* bien qu'ils désignent la même image, ont des sens différents en Anglais. Le premier désigne la viande/animal et le second désigne l'animal *mouton* uniquement. Contrairement au Français le mot *mouton* à la même signification mais pas la même valeur. Quant au linguiste Français Joseph Vendryes (1875-1960), la variété des procédés morphologiques fait que la définition du mot varie suivant les langues. S'il y a des langues où le mot se laisse définir aisément comme une unité indépendante et insécable, il en est d'autres où il se fond en quelque sorte dans le corps de la phrase, où l'on ne peut à vrai dire le définir qu'à condition d'y englober une masse d'éléments variés. La difficulté à instituer le mot comme concept opératoire a conduit de

---

<sup>4</sup> D'après le centre national de ressources textuelles et lexicales. Cette définition est attribué à (Proust, Chron., 1922, p.107).

<sup>5</sup> Forme minimum douée de sens, libre ou liée à une autre forme.

<sup>6</sup> D'après Ferdinand de Saussure, linguiste Suisse, décédé en 1913. Alors que ces élèves ont publié ses cours de linguistique générale en 1916.

nombreux linguistes à préférer les concepts désignés par les termes de: syntagme\*, lexie\* ou synthème\*, ces deux derniers étant les plus fréquents avec cette valeur.

Considérons l'exemple suivant :"Le système de surveillance est dédié à l'assistance et le contrôle de l'habitat d'une personne vivant seule, nommée John, détecte la présence d'une personne dans le séjour à instant t0. Il s'agit de John qui se trouve devant sa télévision dans le séjour. Le système détecte à cet instant, la présence d'une personne dans la cuisine. Le système prévient alors la fille de John. ".

En lisant ce dernier paragraphe, on comprend rapidement, qu'une autre personne se trouve au moment dans l'appartement de John. Or que John est censé être seule. Un humain peut facilement aboutir à cette conclusion grâce aux termes soulignés dans le texte. La question est, une machine peut-elle aboutir au même résultat (contexte) ?

Une autre problématique à laquelle nous devons donner une grande importance est la ponctuation. En effet, des mots peuvent contenir des ponctuations que l'on peut supposer ne faisant pas partie du mot. Mais ce n'est pas toujours vrai. De même, le tiret, le point qui peut être utilisé dans des chiffres, des adresses IP, etc. De plus, vouloir lemmatiser un terme en cherchant parfois son infinitif, remet en cause le sens réel d'une phrase. Prenons l'exemple d'un compte rendu médical : « *Le patient prenait le traitement X* ». Cela laisse entendre que ce patient ne prend plus ce traitement. Cette interprétation est possible du fait que le verbe *prendre* peut signifier avaler/manger et de plus est conjugué à l'imparfait (forme du passé). Observons de près l'exemple suivant : « *Épuisé, le patient est resté dans son lit, il lui ainsi impossible d'appeler l'hôpital ou actionner le bouton d'urgence* ». Dans ce texte, la négation est exprimée autrement et la cause de cette incapacité d'agir est explicitée par le mot épuisé. Ce dernier fait partie des morphèmes lexicaux (i.e. lexème), c'est-à-dire, le mot est porteur de sens. Je rappelle qu'une lettre comme s, x, etc. est considérée également comme morphème, si elle participe à former le pluriel des mots. Toutefois, une information telle qu'hôpital dans l'exemple ci-dessus, devrait contribuer à cerner le sens de ce texte. En traitement de langage naturel, nous parlons d'entité nommées. Ces dernières ne doivent pas être lemmatisées ou leur faire appliquer le stemming (racinisation). Je reviendrai sur ce sujet dans la seconde partie de ce document.

Notons au passage que la reconnaissance des entités nommées est aujourd’hui un domaine de recherche très actif. Je tiens à rappeler également que d’après Sylvie Pollastri, dans son livre *Français. Langue et culture*, revient sur la signification d’un sème, en précisant que le sens ou la signification d’un mot n’est pas la réalité qu’il désigne, mais l’idée que l’on se fait de cette réalité. Sylvie s’appuie sur les travaux de B. Pottierds parus en 1963, quand-t-il invoque le lexème fauteuil. *Pottierds* considère que les sèmes pertinents sont: « destiné à ce qu’on s'y assoie », « avec dossier », « pour une personne », « avec bras », « sur pied(s) ». L’unité sémantique de base est le sème, élément de signification minimal, qui n’apparaîtra comme tel qu’en relation avec un autre élément qui n’est pas lui: il n’a de fonction que différentielle et, de ce fait ne peut être saisi que dans un ensemble organique, dans le cadre d’une structure [5].

- **Syntagme** : est un groupe de mots (groupe de morphèmes) qui forme une unité par son sens et par sa fonction, à l’intérieur de la phrase.

-**Lexie** : peut être un mot simple (lexème) ou un groupe de mots.

-**Synthème** : énoncé minimal formé de plusieurs monèmes (unité lexical minimale) et pouvant être analysé en deux ou plusieurs unités de sens.

## Racinisation (stemming, en Anglais)

Ce traitement est une opération ayant pour objectif de supprimer les préfixes et les suffixes des flexions. L’objectif est de générer des racines (radical, stem en Anglais) (nous parlons de dé-suffixation en Français). Mais, il faut être très prudent qu’au résultat obtenu après la racinisation. En effet, le mot engendré peut cependant ne pas exciter dans le dictionnaire de la langue. Contrairement au principe de lemmatisation, le lemm résultant est forcément un mot de la langue. Les outils de racinisation (i.e. les algorithmes) les plus connus sont développés par Lovins, Porter et Paice [6][7][8]. Quant à la langue Française, l’algorithme utilisé est CARY qui s’appuie sur la l’algorithme Porter. Cary a été proposé en 2002 par M. Paternostre et al.[9]. Tous ces algorithmes sont souvent utilisés dans la recherche d’information, la reconnaissance d’entité nommée ou même la traduction automatique. En effet, réduire des mots à leur

racine permet de conserver une seule forme d'un mot en partageant la même racine, mais n'ont pas forcément la même portée sémantique. Pour plus de détails sur le principe de racinisation (stemming) l'étudiant est appelé à lire ces documents accessibles sur le net<sup>78</sup>.

En résumé, ces algorithmes procèdent en plusieurs étapes, et parmi eux, ceux qui utilisent des dictionnaires. Les approches à base de dictionnaire sont plus longues mais en l'avantage de ne pas faire d'erreur sur des mots déjà listés. Quant à ceux qui ne sont pas basés sur des dictionnaires, auront l'avantage d'être rapide, mais le taux d'erreur est élevé. L'algorithme de Porter est fondé entre autres sur cinq règles de contexte nécessaires pour décider si oui ou non un suffixe devrait être supprimé. Par exemple, La terminaison en -ing, par exemple, ne sera enlevée que si le radical comporte au moins une voyelle. De cette manière, "troubling" deviendra "troubl", nous l'avons vu dans l'exemple exécuté avec le langage R, alors que "sing" restera "sing". Quant à l'algorithme Carry, utilise également des règles et des conditions de désuffixation, ainsi quand l'analyseur reconnaît un suffixe de la liste, soit il le supprime, soit il le transforme. Par conséquent, la racinisation est un traitement à faire en dernier.

## 1. Utilisation du package textstem.

Le tableau suivant présente les fonctions de la librairie textstem.

Fonction	Tâche	Description
stem_words	stemming	Stem words
stem_strings	stemming	Stem strings
lemmatize_words	lemmatizing	Lemmatize words
lemmatize_strings	lemmatizing	Lemmatize strings
make_lemma_dictionary_words	lemmatizing	Génère un dictionnaire de lemme pour un texte

---

<sup>7</sup> <http://www.otlet-institute.org/docs/Carry.pdf>

<sup>8</sup> <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.848.7219&rep=rep1&type=pdf>

Installer la librairie *textstem* si ce n'est pas encore fait, et la librairie *korpus*. Ensuite charger ces deux librairies. Dans le cas un ou plusieurs packages sont nécessaires pour la compilation, alors il faut les installer comme expliquer au début de ce document. N'oublier pas de charger les modules (packages) avec la commande library. La librairie korpus a été initié depuis 2011 dans l'objectif de calculer la similarité entre des documents. Son succès a permis d'avoir un package pour R avec des formules et des enveloppeurs (Wrappers) pour des bases de données de corpus linguistiques et POS tagger. Pour plus de détails sur la librairie korpus, je vous invite à lire le document disponible en ligne :

[https://cran.r-project.org/web/packages/koRpus/koRpus.pdf\).](https://cran.r-project.org/web/packages/koRpus/koRpus.pdf)

Une fois que tout se déroule bien. Exécute la commande pour faire le stemming

```

library(textstem)
par<-c("created", "stopped", "driven", "broken", "drove", "wings", "lazy", "studio",
      "studious", "many", "policy", "policing", "speaks", "spoke", "teach",
      "educator", "educational", "educated", "teacher", "manager", "mange",
      "laziness", "best", "winner", "board", "boarding", "growing", "wine", "won", "wrote");
stem(par)
stem_words(x, language = "porter", ...)
```

Values

Original Word	Stemmed Word
created	creat
stopped	stop
driven	drive
broken	break
wings	wing
lazy	lazy
studio	studiou
studious	studious
many	man
policy	polici
policing	policin
speaks	speak
spoke	spok
teach	teach
educator	educator
educational	educational
educated	educat
teacher	teacher
manager	mange
mange	mange
laziness	laziness
best	best
winner	winner
board	board
boarding	boarding
growing	grow
wine	wine
won	won
wrote	wrote

Remarquer que des verbes ne sont pas tronqués.

Exécuter les commandes (instructions) suivantes

```

19 | library(syllly)
20 | library(koRpus.lang.en)
21 | library(koRpus)
22 | library(syllly.fr)
23 | library(textstem)
24
```

Commencer par afficher la liste des mots de l'exemple introduit ci-dessus. Pour se faire, il suffit de saisir le nom de votre variable, en l'occurrence "*par*", ensuite ré-exécuter le stemming, comme suit :

```

11
12
13
14 wordStem(c("données","issues","maux", "mal",
15           "mise","miser","mettre", "ligne", "ligner","bas","base", "contrat","contracter", "sont", "peu","peut", "fiables")
16
17
18
19 #Le script suivant est pour la lemmatization
20 # Pour ce faire je vais utiliser la bibliothèque textstem
21 library(syll)
22 library(koRpus.lang.en)
23 library(koRpus)
24 library(syll.fr)
25 library(textstem)
26 par
27 wordStem(par)
28
28:1 (Top Level) :

```

Console Terminal ~

```

> par
[1] "created"    "scopped"     "driven"      "broken"      "drove"       "wings"       "lazy"        "studio"      "studious"    "many"        "policy"
[12] "policing"   "speak"       "spoke"       "teach"       "educator"    "educational" "educated"    "teacher"     "manager"    "manage"      "laziness"
[23] "test"        "Winner"      "board"       "boarding"    "growing"     "wine"        "won"         "wrote"
> wordStem(par)
[1] "creat"      "stop"       "driven"     "broken"     "drove"      "wing"       "lazi"        "studio"     "mani"       "polici"     "polic"
[16] "educ"       "educ"       "teacher"    "manag"      "lazi"       "best"       "winner"     "board"      "board"      "grow"       "wine"       "won"        "wrote"
>

```

Pour celles ou ceux qui ne l'ont pas encore fait, notez vos remarques quant au résultat du stemming.

Dans ce qui suit, je suppose que TreeTagger est installé. Avant de continuer, il est possible que TreeTagger ne fonctionne pas avec R, une des solutions possible consiste à renommer le fichier french-abbreviations dans le répertoire TreeTagger\lib en french-abbreviations-utf8

Les entrées des commandes sont des fichiers textes en format UTF-8. Et grâce à la commande `getTransformations()`, vous pouvez visualiser les opérations utiles à appliquer sur des textes.

### Exemples.

Exécuter dans R les commandes suivantes :

```

library(tm) library(tidyverse) library(textstem) library(koRpus)
library(koRpus.lang.en) library(syll)

```

Pour les valeurs de la variable `-par-`, vous pouvez utiliser d'autres mots que ceux utilisé dans mon exemple.

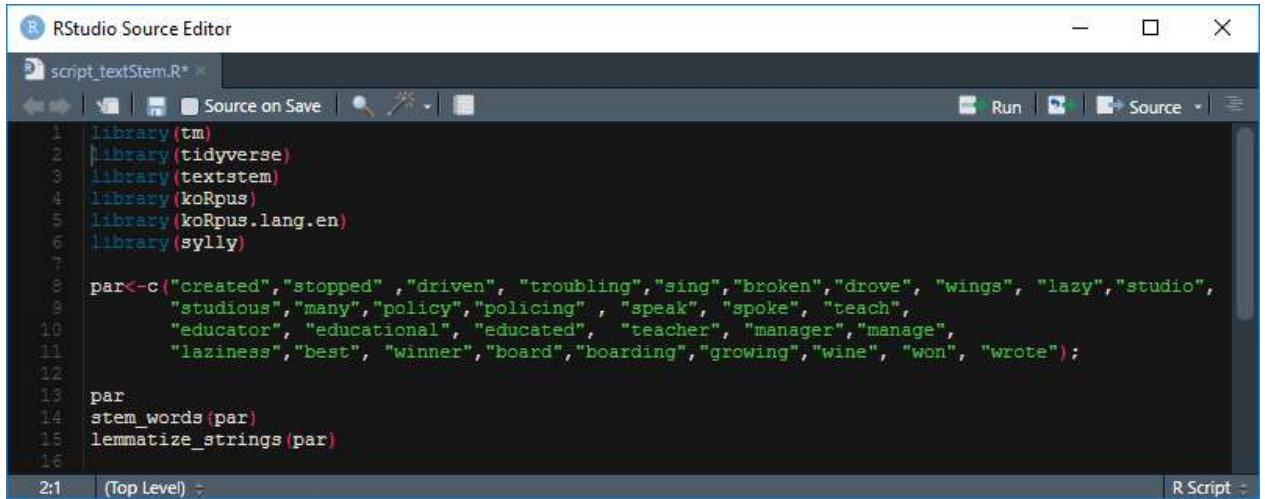
```

par<-c("created","stopped" , "driven", "troubling","sing","broken","drove", "wings",
"lazy","studio", "studious","many","policy","policing" , "speak", "spoke", "teach",

```

```
"educator", "educational", "educated", "teacher", "manager", "manage",
"laziness", "best", "winner", "board", "boarding", "growing", "wine", "won", "wrote");
```

- stem\_words(par)
- lemmatize\_strings(par)



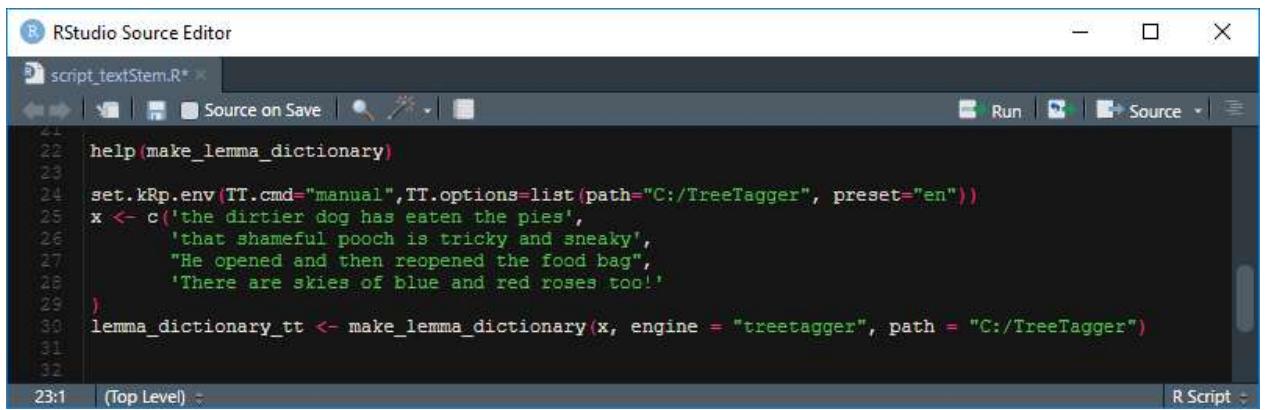
The screenshot shows the RStudio Source Editor window with the script `script_textStem.R*`. The code is as follows:

```
library(tm)
library(tidyverse)
library(textstem)
library(koRpus)
library(koRpus.lang.en)
library(syll)

par<-c("created", "stopped", "driven", "troubling", "sing", "broken", "drove", "wings", "lazy", "studio",
      "studious", "many", "policy", "policing", "speak", "spoke", "teach",
      "educator", "educational", "educated", "teacher", "manager", "manage",
      "laziness", "best", "winner", "board", "boarding", "growing", "wine", "won", "wrote");

par
stem_words(par)
lemmatize_strings(par)
```

Pour utiliser le lemmatiseur avec treetagger, il suffit d'exécuter les commandes suivantes. Attention, comme indiqué dans les paramètres de cette commande, le chemin (path) de treetagger est par défaut le C. Je vous propose d'exécuter la commande : `help(make_lemma_dictionary)` pour lire la documentation.



The screenshot shows the RStudio Source Editor window with the script `script_textStem.R*`. The code is as follows:

```
help(make_lemma_dictionary)

set.kRp.env( TT.cmd="manual", TT.options=list(path="C:/TreeTagger", preset="en"))
x <- c('the dirtier dog has eaten the pies',
       'that shameful pooch is tricky and sneaky',
       'He opened and then reopened the food bag',
       'There are skies of blue and red roses too!')
lemma_dictionary_tt <- make_lemma_dictionary(x, engine = "treetagger", path = "C:/TreeTagger")
```

Afin de voir l'effet de la catégorisation grammatical avec treetagger. Je vous invite à créer un fichier nommé `text2.txt` avec l'éditeur bloc note dans un répertoire dédié, copier le texte à partir du tableau suivant, et coller-le dans ce fichier.

The TreeTagger is a tool for annotating text with part-of-speech and lemma information. It was developed by Helmut Schmid in the TC project at the Institute for Computational Linguistics of the University of Stuttgart.

Ensuite, avec la fonction `setwd( )`, commencer par indiquer le répertoire où vous avez créé ce fichier. Charger les librairies `library(koRpus)` , `library(koRpus.lang.fr)`. Ensuite exécuter la commande suivante :

```
tagged.text <- treetag(  
  "text2.txt", treetagger="manual", lang="en", debug = TRUE,  
  TT.options=list( path="C:/treetagger/", preset="en" ),  
  doc_id="firstText"  
)
```

Pour visualiser les résultat, exécuter la commande `tagged.text`, ensuite `summary(tagged.text)`.

## Manipulation avec R

Dans ce qui suit, je suppose que vous avez récupérer les fichiers de travail et que vous les avez mis dans un répertoire nommé `docs`. Ainsi les commandes suivantes vont respectivement indiquer le chemin vers les données de travail et la seconde crée un corpus de ces fichiers. Ce corpus est représenté par la variable `mydocs`. Vous devez donc indiquer le chemin qui est le vôtre et vous avez également la possibilité de modifier le nom de la variable de votre corpus.

```
> setwd ("E:\\TWM\\Documents\\\\docs")  
> mydocs<-Corpus(DirSource(getwd()))  
  
library(tokenizers)  
wordsT1<- tokenize_words(mydocs[1])  
length(wordsT1[[1]])
```

```

tablefichier1 <- table(wordsT1[[1]])
library(tidyverse)
tablefichier_V1 <- tibble(word = names(tablefichier1), count
= as.numeric(tablefichier1))
arrange(tablefichier_V1, desc(count))

```

Commencer par charger la librairie tm et tokenizers.

Vous allez essayer de lire le contenu d'un fichier en utilisant la fonction *inspect* comme suit : `inspect(mydocs[1])`. Ensuite lister l'ensemble des token (mots ou de phrases) de votre texte. Sur la console, vérifier que l'opération de la tokenization s'est bien déroulée.

```

tokenize_words(mydocs[1])
tokenize_sentences(mydocs[1])

```

Le principe de tokenisation est appliqué en vue de divisés les documents en phrase, mots, syllabe ou même un phonème (la lettre s marque de pluriel est un phonème dans le mot humain (s)). Une fois la liste de *token* créée, vous pouvez faire des tests de statistiques pouvant être utiles telles que calculer la fréquence d'un mot dans un document, le nombre de mots, afficher dans un ordre précis la liste et le nombre d'occurrence de chaque mot, etc. Je vous invite à exécuter les fonctions suivantes et à observer le résultat sur la console. Remarquer l'utilisation de librairie *tidyverse* <https://www.tidyverse.org/packages/>. Il faut installer cette librairie si ce n'est pas déjà fait.

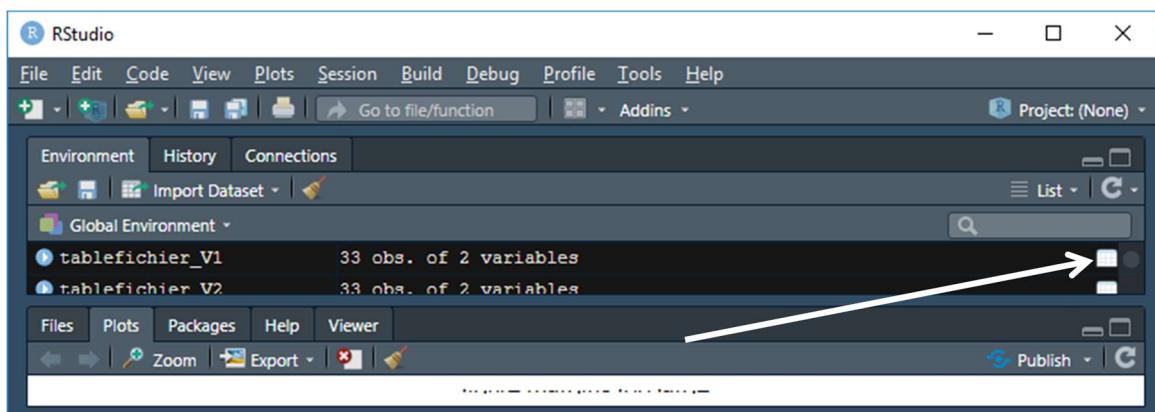
```

library(tokenizers)
wordsfichier1<- tokenize_words(mydocs[1])
length(wordsT1[[1]])
tablefichier1 <- table(wordsfichier1[[1]])
tablefichier1
library(tidyverse)
tablefichier_V1 <- tibble(word = names(tablefichier1), count = as.numeric(tablefichier1))
arrange(tablefichier_V1, desc(count))

```

Parmi les fonctions utilisées, *tibble*<sup>9</sup> qui est préféré dans plusieurs cas au *data.frame* de R. Entre autres, *tibble* ne change jamais le type des entrées (par exemple il ne convertit jamais les chaînes en facteurs), il ne change jamais les noms des variables, les noms de colonne peuvent ne pas commencer par une lettre ou contenir des caractères inhabituels comme un espace. Pour faire référence à ces variables, vous devez les entourer de backticks (<https://r4ds.had.co.nz/tibbles.html>).

Une manière de visualiser le contenu d'une structure de données R est de faire un clic sur l'icône référençant la variable dans l'onglet environment de RStudio, voir figure suivante.



```
> wordsT1<-tokenize_words(mydocs[1])
> wordsT1
$`a good health.txt`
[1] "people"      "who"        "exercise"     "live"       "longer"      "and"        "keep"        "well"
[9] "threi"        "health"      "on"          "average"    "than"        "those"      "who"        "don't"
[17] "dozens"       "of"          "studies"     "show"       "that"        "regular"    "physical"   "activity"
[25] "lowers"        "your"        "risk"        "of"          "heart"      "disease"    "stroke"     "diabetes"
[33] "and"          "avoid"       "other"       "health"     "problems"

$<NA>
[1] "list"        "language"   "en"

$<NA>
[1] "list"
```

---

<sup>9</sup> <https://cran.r-project.org/web/packages/tibble/vignettes/tibble.html>

Comme je l'ai exposé au début du document, le besoin d'un côté d'explorer et d'analyser la quantité importante de données textuelles et grâce à l'évolution technologique d'un autre côté, fait que le traitement et l'analyse textuelle se compte en seconde. Ce qui explique en grande partie la raison pour laquelle la recherche, et les innovations s'activent dans le traitement des corpus de textes. Dans le domaine de fouilles de textes, de nombreuses applications telles que, le filtrage des e-mails, l'extraction de thèmes à partir de page web, l'analyse des comptes rendus médicaux, s'appuient sur des méthodes utilisées en Data Mining (fouilles de données), dites méthodes statistiques ou probabilistes, et d'autres reposent sur des méthodes linguistiques. Contrairement aux méthodes linguistiques, les méthodes statistiques ne s'intéressent pas à la sémantique. En vue d'utiliser les techniques de data mining, et faciliter la manipulation des documents, il est donc indispensable de transformer les données non structurés existant dans des documents en des structures décrivant ces documents. En effet, ces documents seront représentés par des vecteurs dans un espace nommé modèle d'espace vectoriel (Vector Space Model) contenant souvent la fréquence d'occurrences des mots dans le corpus. Ces vecteurs sont nommés des vecteurs de caractéristiques (features) nommé poids du terme. Un feature désigne une entité sans structure interne (i.e. une dimension dans l'espace d'entité). Ces vecteurs sont rangés en une matrice, dite matrice termes-documents (document-term matrix) et parfois matrice sac de mots (bags of words, en Anglais). Dans ce type de matrice, les colonnes représentent les mots, quant aux lignes représentent les documents. Par conséquent, une cellule ( $t_i, d_i$ ) de la matrice est une entrée indiquant la fréquence d'un terme  $t_i$  d'un document  $d_i$ , nous parlons d'attributs-valeurs d'une matrice.

L'utilisation de vecteur de mots est apparue dans les années 75 [11] qui consiste à soumettre un corpus à un processus de prétraitement, en vue de réduire la dimension de la matrice. L'inconvénient majeur est que l'ordre des mots n'est pas important, ainsi, on ne différencie pas entre les expressions telles que : l'enfant mange une pomme ou une pomme mange un enfant. Les ponctuations sont supprimées, l'information grammaticale, comme les synonymes d'un mot ne sont pas pris en compte. De plus, le nombre de colonnes augmente considérablement, le contexte des mots dans les phrases disparait complètement, etc. Ce qui pose le problème de perte

d'information. La perte d'information peut se produire également suite aux nombreuses opérations de nettoyage d'un corpus (lemmatisation, stemming, supprimer stops words, etc.). Enfin, lors des étapes de prétraitement, des mots considérés comme non pertinent (i.e. ne permet pas de différencier entre document) ce qui fait que ces mots sont supprimés. Ces mots sont communément nommés mots vide (stop words, en Anglais). Comme exemple de mots vide, nous pouvons citer : What, Who, vive, vis-à-vis, instant, à, cet, etc. les méthodes qui pratique la suppression des mots vides considèrent que ces mots n'ont pas de valeur ajoutée, or dans nos exemples précédents, ce raisonnement est loin d'être vrai. Penser à l'exemple de John où une analyse du texte doit conclure que John n'est pas seul dans sa maison. Toutefois, la suppression de ces mots vides offre bel et bien des améliorations en raison de la réduction de la matrice.

Bien que le sens permettant d'interpréter un texte comme un humain est définitivement écarté; cependant, les résultats obtenus par les algorithmes utilisant ces matrices continuent à donner des résultats acceptables.

Grouper des documents similaires permet d'accélérer le processus de recherche. En effet, les documents les plus similaires (cohérents), ont plus de probabilité d'être pertinents si un des documents de la même classe est considéré comme une réponse à une requête donnée. De plus, le clustering est un processus non supervisé par lequel les objets sont classés en groupes appelés clusters. Contrairement à la catégorisation, il s'agit de grouper les textes en clusters significatifs sans aucune information disponible au préalable. Ce qui nous emmène donc à parler de la similarité des contenus des documents. Par conséquent, il est important de définir une mesure de similarité. Dans la littérature, de nombreuses mesures géométriques de similarité, telle que mesure de distance euclidienne, manhattan, cosinus, etc.

Avant d'appliquer des mesures de similarité, un des traitements primordiaux les plus intuitifs consiste à calculer la fréquence d'apparition d'un mot dans un document. Nous parlons dans ce cas de méthodes de pondération des termes (feature). Il s'agit de déterminer le poids de chaque terme. Chaque vecteur représentant un document dans cet espace aura un composant pour chaque mot. La manière la plus simple est la

pondération binaire. Il s'agit ici d'attribuer un poids de valeur 1 (valeur positive) si le mot correspondant est présent dans le document. Si un mot n'est pas présent dans le document, la composante du mot du vecteur de document sera nulle. Bien que cette approche soit simple, elle présente l'inconvénient majeur d'attribuer la même importance à tous les termes (e.g. un terme qui apparaît une seule fois a le même poids qu'un terme qui apparaît 100 fois). Afin de contourner cet inconvénient, des schémas de pondération plus complexes peuvent être utilisés. Il s'agit de tenir compte des fréquences du mot dans le document, dans la catégorie et dans l'ensemble de la collection (i.e. corpus). On parle alors de TF (term frequency). Cette dernière donne un aperçu sur le poids d'un terme dans un document par rapport à l'ensemble du corpus. Par conséquent, le nombre réel d'apparition d'un terme est comptabilisé. Mais, cette approche présente comme nous allons le voir plus tard, le désavantage conséquent si utilisée avec une distance Euclide et que les documents n'ont pas tous presque la même longueur. De plus, avec cette approche, la prédominance des termes dans le corpus est écartée. Une des solutions à ce dernier inconvénient consiste à faire privilégier les termes rares dans un corpus et qui apparaissent dans un document. En effet, un terme qui apparaît dans tous les documents, ne permet pas de différencier entre ces documents. C'est pourquoi, une variante à cette approche de TF consiste à utiliser Inverse document frequency (IDF) pour faire croître le poids d'un terme. On sait que les termes qui apparaissent fréquemment dans les environnements de texte locaux, tels que des paragraphes de texte particuliers, mais relativement rares dans la collection de texte complète, sont importants à des fins de récupération de texte. Cela se reflète dans la fonction de pondération des termes bien connue  $tf \cdot idf$  (term frequency times inverse collection frequency), qui préfère les termes avec des fréquences d'occurrence locales élevées [12-13]. Pour l'auteur, il considère que des situations où les textes disponibles couvrent une variété de sujets différents, les méthodes d'analyse de texte basées sur des concepts qui sont souvent préconisées dans la littérature ne sont pas utilisables, car il est impossible de construire les bases de connaissances qui spécifient le contenu et la structure des sujets d'intérêt. Au lieu de cela, le texte lui-même doit constituer la base des opérations d'analyse de texte: les termes de contenu attachés aux textes doivent être obtenus à partir des textes par une

opération d'indexation automatique, et les listes de vocabulaire et les thésaurus préconstruits qui sont difficiles à fournir doivent être utilisés avec parcimonie, ou pas du tout. De plus, l'auteur dans [10] montre que des preuves expérimentales suggèrent que l'utilisation uniquement des 10% des mots les plus fréquents ne réduit pas les performances des classificateurs. Cela semble contredire la «loi» bien connue de l'IR, selon laquelle les termes avec une fréquence de document faible en moyenne sont les plus informatifs. Un autre calcul consiste à chercher donc la fréquence d'un mot dans l'ensemble du corpus (TF.IDF= Term Frequency-Inverse Document Frequency). Le poids TF.IDF mesure l'importance d'un mot en fonction de sa fréquence dans le document pondéré par la fréquence d'apparition du terme dans tout le corpus. Ainsi, à chaque fois que la valeur tf-idf est élevée, signifie que le terme correspondant est important dans un document et rare dans les autres documents.

### **Exemple de calcul de normalisation.**

Pour faire simple considérons trois documents représentés par D1, D2, D3 et trois termes T1,T2 et T3, tableau suivant. Ce tableau présente un aperçu sur l'importance d'un terme dans un document (prise en compte de la répétition des mots). Nous parlons de Term Frequency (TF)

	T1	T2	T3
D1	1	7	1
D2	8	1	0
D3	3	5	0

Afin de diminuer les écarts causés par TF et surtout d'en tenir compte de la longueur des documents, une des possibilités est d'utilisé une normalisation logarithmique.  $TF_{t,d} = \begin{cases} 0 & si f_{t,d} = 0 \\ 1 + \log_{10} f_{t,d} & \end{cases}$  ou une normalisation simple  $TF_{t,d} = \frac{f_{t,d}}{\sum_t f_{t,d}}$

Appliquer aux documents D1,D2 et D3 ces deux formules aboutissent à la réduction suivante.

	T1	T2	T3
D1	1	1.84	1
D2	1.90	1	0
D3	1.47	1.69	0

	T1	T2	T3
D1	0.11	0.77	0.11
D2	0.42	0.05	0
D3	0.38	0.63	0

### Pondération avec IDF.

La formule à appliquer est la suivante :  $IDF_{t,d} = \log_{10} \frac{DCS}{D_t}$ .

$DCS$  est l'ensemble des documents, quant à la variable  $D_t$  représente le nombre de documents où  $t$  apparaît. Bien que notre exemple est simple, mais cela à permis de vérifier que les termes T1 et T2 ne peuvent pas être utilisés pour différencier entre les documents (i.e. l'importance du terme dans le corpus). Il faut souligner que l'occurrence élevée d'un terme permet de dégager le domaine d'un document.

Afin de donner des poids aux termes les moins fréquents puisqu'ils sont les plus discriminants. Il s'agit d'utiliser la formule suivante :  $TFIDF_{t,d} = f_{t,d} * idf_t$ . Remarquons que la valeur TFIDF de T3 est élevée.

	T1	T2	T3
D1	1	7	1
D2	8	1	0
D3	3	5	0
IDF	<b>0</b>	<b>0</b>	<b>0.47</b>

	T1	T2	T3
D1	0	0	<b>0.47</b>
D2	0	0	0
D3	0	0	0

Pour éviter d'avoir une valeur nulle pour IDF, un 1 est ajouté au calcul de log. Comme vous pouvez calculer TF en considérant *le nombre d'occurrence de terme / la somme total des termes document*.

### Manipulation avec R.

Exécuter les instructions suivantes une à une et dans l'ordre.

```
> matrice = as.matrix(data.frame(c(1,8,3), c(7,1,5), c(1,0,0)))
> idfdoc<-function(col) {
```

```

c.s<-length(col)
doc.count<-length(which(col>0))
log10(c.s/doc.count)
}

> tfidf<-function(x,idf) {
  x*idf
}

> idfmatrice<-apply(matrice,2,idfdoc)
> tfidfm<-apply(matrice,1,tfidf,idf=idfmatrice)
> t(tfidfm)

```

La dernière commande affiche la valeur tfidf de la matrice. La fonction t permet d'inverser la matrice. Essayer de visualiser la matrice sans utiliser la fonction t pour comprendre.

### **Exercice.**

Utiliser la fonction suivante pour calculer tfidf en considérant *le nombre d'occurrence de terme / la somme total des termes document.*

```

> t.f<-function(row) {
  row/sum(row)
}
tfmat<-apply(matrice,1,t.f)

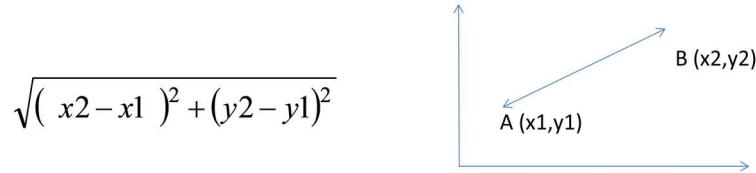
```

## **Mesures de similarité**

Plusieurs méthodes de mesures existent. L'utilisation d'une des méthodes dépend des caractéristiques des données. Dans un espace vectoriel, le calcul de similarité est basé sur le calcul de la distance entre vecteurs. La méthode par défaut est celle d'Euclide, mais il existe de nombreuses autres méthodes telles que : cosine, maximum, manhattan, minkowski, pearson, etc. Mais, attention, le choix de mesure de distance est une étape importante puisqu'elle influence le résultat final.

Dans un modèle d'espace vectoriel, la fonction de similarité est généralement basée sur la distance entre les vecteurs dans une certaine métrique. Parmi les mesures de similarité les plus populaires est la distance Euclide.

Pour simplifier la compréhension, je vais reprendre l'exemple de la matrice des trois documents et trois termes. Je rappelle que la distance euclidienne est utile quand les documents ont presque la même longueur



La distance euclidienne découle de la généralisation du théorème de Pythagore. Ainsi, la distance entre deux éléments revient à calculer l'hypoténuse. Deux éléments identiques auront donc une distance nulle. Pour les documents, la formule à appliquer à deux documents est :  $d(u, v) = \sqrt{\sum_{j=1}^t (u_i - v_j)^2}$  où t est le nombre de termes.

## Manipulation avec R

Reprendons toujours la même matrice.

```
> matrice
  c.1..8..3. c.7..1..5. c.1..0...
[1,]      1      7      1
[2,]      8      1      0
[3,]      3      5      0

> dist(matrice, method = "euclidean")
     1       2
2 9.273618
3 3.000000 6.403124

>
```

Pour la première valeur  $d(D1, D2) = \sqrt{(8 - 1)^2 + (1 - 7)^2 + (0 - 1)^2}$ . Ainsi, la valeur entre  $d(D1, D1)=0$ . Cette valeur n'est pas affichée.

## Exercice.

Appliquer l'algorithme Euclide à la matrice pondérée.

Similarité cosine, contrairement à Euclide, la mesure cosine s'appuie sur l'angle entre deux vecteurs. La similarité est d'autant plus pertinente si l'angle est petit. Remarquons que les résultats sont dans l'intervalle 0 et 1. Et puisqu'elle ne s'intéresse uniquement aux occurrence, cosine permet de comparer des documents de différentes tailles.

$$\cos(d, c) = \frac{\sum_{i=1}^n d_i c_i}{\sqrt{\sum_{i=1}^n d_i^2} \sqrt{\sum_{i=1}^n c_i^2}}$$

	T1	T2	T3
D1	1	7	1
D2	8	1	0
D3	3	5	0

$$\cos(d1, d2) = \frac{1*8+7*1+1*0}{\sqrt{1+49+1}\sqrt{64+1+0}} = \cos(d2, d3) = \frac{3*8+5*1+0*0}{\sqrt{64+1+0}\sqrt{9+25+0}} =$$

## Manipulation avec R.

Pour utiliser la mesure de similarité cosine, nous allons utiliser la librairie quanteda<sup>10</sup>. Pour ce faire, installer la librairie quanteda si ce n'est pas encore fait. Ensuite charger cette librairie. Enfin, exécuter les commandes suivantes et comparer vos résultats effectués manuellement.

**Question :** Analyser le résultat. Remarquer que le choix de pondération est le nombre d'occurrences des termes.

```
> library(quanteda)  
> matrice_dfm<-quanteda::as.dfm(matrice)
```

---

<sup>10</sup> <https://cran.r-project.org/web/packages/quanteda/index.html>

```

> matrice_dfm
> resultat_cosine<-textstat_simil(matrice_dfm, min_simil = 0.2, margin
= c("documents", "features"), method = "cosine")
# deux méthodes pour afficher du calcul.
>as.list(resultat_cosine)
>as.list(resultat_cosine, diag = FALSE)

```

## Catégorisation et classification de documents

Pour être regroupés, les documents doivent être convertis en vecteurs dans l'espace d'entité. La représentation de document en sac de mots, et l'utilisation des différentes possibilités de pondération suppose alors que chaque mot est une dimension dans l'espace des fonctionnalités. Un des problèmes les plus importants consiste donc la réduction de la dimensionnalité des vecteurs. Une des méthodes qui a émergé consiste à utiliser l'indexation sémantique latente. Cette approche sera détaillée dans d'autres sections. Attention à ne pas confondre entre l'extraction d'information (IE : Information Extraction) et la recherche d'information. Pour cette dernière, il s'agit de trouver des documents recherchés en privilégiant la validation manuelle de la pertinence de ces documents. Quant à IE a pour rôle d'identifier des connaissances pertinentes et à les représenter sous un format exploitable par une machine.

Il n'est pas étrange donc que la classification des textes parmi d'autres prédéfinis est un des domaines le plus connu. L'automatisation de la classification remonte au début des années 1960 [14] quand Maron a utilisé le vocabulaire contrôlé pour l'indexation des œuvres scientifiques. Le terme « indexation automatique» désigne le problème de décider de façon mécanique à quelle catégorie (sujet ou domaine de connaissance) appartient un document donné. Il s'agit du problème de décider automatiquement ce qu'est un "document". Pour la catégorisation donc des textes, des approches privilégient d'apprentissage automatique (ML : Machine learning, en Anglais). Dans l'approche ML, le classificateur est construit automatiquement en

apprenant les propriétés des catégories à partir d'un ensemble de documents de formation pré-classifiés. Cette manière d'opérer est dite supervisé, car elle est guidée par l'application de la fonction d'affectation de catégorie vraie connue sur l'ensemble d'apprentissage. Selon Feldman et al. [10], quatre problèmes principaux doivent être pris en compte lors de l'utilisation de techniques d'apprentissage automatique pour développer une application basée sur la catégorisation de textes. Tout d'abord, il faut décider des catégories qui seront utilisées pour classer les instances. Deuxièmement, fournir un ensemble de formation pour chacune des catégories. En règle générale, environ 30 exemples sont nécessaires pour chaque catégorie. Troisièmement, décider des fonctionnalités qui représentent chacune des instances. Habituellement, il est préférable de générer autant de fonctionnalités que possible car la plupart des algorithmes pourront se concentrer uniquement sur les fonctionnalités pertinentes. Enfin, il est nécessaire de décider de l'algorithme à utiliser pour la catégorisation. Parmi ces algorithmes, la logique de régression, naïve bayes, réseaux de neurones, etc. Il existe de nombreuses approches de l'apprentissage des classificateurs; certains d'entre eux sont des variantes d'algorithmes ML plus généraux, et d'autres ont été créés spécifiquement pour la catégorisation.

Les classifieurs probabilistes applique le théorème Bayes tel que la probabilité qu'un document  $d$  appartienne à un corpus  $c$  calcul la formule (1).

$$P(c | d) = \frac{P(d|c)P(c)}{P(d)} \dots \dots \dots \quad (1)$$

$$P(c | d) = \prod_i P(w_i | c) \text{ tel que } d(w_1, w_2, \dots, w_i) \dots \dots \dots \quad (2)$$

Un document est sélectionné et une machine cherche à voir lequel des termes (mots-clés) sélectionnés sont contenus dans ce document. Sur la base de l'occurrence des mots indices, la machine fait une inférence quant à la catégorie de sujet à laquelle appartient le document en question. Ainsi, l'inférence est une transition inverse de la preuve à l'hypothèse; le calcul de probabilité nous fournit le schéma approprié pour calculer les valeurs. Un document  $d$  ne contenant qu'un seul des mots-clés  $w$ , alors cherche la probabilité que  $d$  appartienne à une des catégories de documents  $c_1, c_2, \dots$  revient à calculer sa valeur avec la formule (1). L'application de cette formule suppose

que la structure de document d comme un vecteur de mots dont les coordonnés sont tous indépendants (2). Ce type de classifieur est nommé Naïve Bayes inventé pour la première en 1961 [14]. L'auteur s'appuie sur l'occurrence de certains indices dans un document, pour décider automatiquement à laquelle de nombreuses catégories de sujets appartiennent un document en question. Cette appellation est souvent fausse du fait que l'hypothèse suppose naïvement (de manière simplificatrice) la façon dont les caractéristiques (feature) interagissent.

Quant au clustering, les algorithmes les plus couramment utilisés sont entre autres, K-means, l'algorithme de clustering probabiliste basé sur EM ((expectation maximization (EM) et le HAC (agglomération hiérarchique). Comme la représentation des documents en sacs de mots ne tient pas compte d'un côté des liens sémantique existant entre les termes (i.e. le contexte) et d'un autre côté la manipulation des vecteurs de grandes dimensions. On assiste à l'émergence de nouvelles approches telles que l'apprentissage profond (deep learning, en Anglais) et les réseaux neuronaux. Habituellement, les nœuds d'entrée du réseau reçoivent les valeurs d'entité, les nœuds de sortie produisent les valeurs d'état de catégorisation et les poids de liaison représentent des relations de dépendance. Pour classer un document, ses poids d'entités sont chargés dans les nœuds d'entrée; l'activation des nœuds se propage vers l'avant à travers le réseau, et les valeurs finales sur les nœuds de sortie déterminent les décisions de catégorisation [10]. Le type de réseau neuronal le plus simple est un perceptron. Il n'a que deux couches - les nœuds d'entrée et de sortie. Un tel réseau équivaut à un classificateur linéaire. Les réseaux plus complexes contiennent une ou plusieurs couches cachées entre les couches d'entrée et de sortie. Cependant, les expériences ont montré une très faible - ou aucune – amélioration des réseaux non linéaires sur leurs homologues linéaires dans la tâche de catégorisation de texte [15-16-17]. La prise en compte des dépendances temporelles dans les données séquentielles devient possible avec les modèles de réseaux neuronaux récurrents. Cette approche sera étudiée dans un autre document faisant suite à celui-ci.

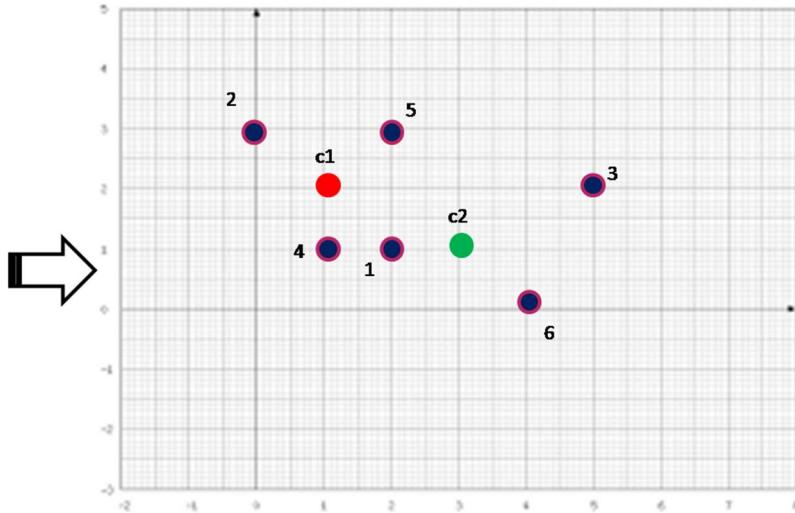
## Création de cluster (classe) avec K-mean.

Il s'agit d'une méthode de machine learning non supervisé. Cette méthode permet regrouper des éléments tels que des thèmes, documents, images, pages web, en K clusters (classe) distincts qui ne se chevauchent pas. Le nombre de cluster K doit être spécifié au début du lancement de l'algorithme, mais des approches qui seront expliquées lors de la manipulation de cet algorithme avec R existent pour optimiser le choix du nombre de clusters. Contrairement à HCA (voir section suivante), à l'intérieur de chaque cluster, aucune mesure d'hiérarchisation entre document n'est utilisée, elle repose cependant sur la minimisation de la distance entre un élément et le centroïde du cluster (i.e. valeur intra-cluster) auquel il appartient. Avec K-means, dans chaque groupe, les données sont les plus similaires possibles (la valeur intra-classe est la plus faible possible) et la valeur interclasse entre groupe est le plus grand possible. Le principe de cet algorithme se résume en :

1. Définir au hasard k centres de gravité pour chaque cluster. Ces centres doivent être choisis en sorte à différencier les clusters. La meilleure astuce est de les éloigner les uns des autres. Notons que la classification des individus dépend donc du choix des centres initiaux.
2. Calculer la distance entre les individus et les k centres choisi lors de la première étape et associer chaque individu au centre de gravité (i.e. barycentres) le plus proche.
3. Recalculer k nouveaux centres de gravité des groupes issus de la dernière opération.
4. Refaire les deux opérations 2 et 3 jusqu'à ce que les barycentres se stabilisent, si un nombre d'itération est atteint ou soit l'inertie interclasse (distance entre les différents barycentres ne s'améliore pas trop).

## Exemple de K-means

	t1	t2
Texte1	2	1
Texte2	0	3
Texte3	5	2
Texte4	1	1
Texte5	2	3
Texte6	4	0



Pour faire simple, considérons cinq textes et deux termes  $t_1$  et  $t_2$ . La matrice des termes documents est représentée par la figure ci-dessus. Supposons le choix aléatoire de deux centres  $c_1(1,2)$  et  $c_2(3,1)$ . La matrice montre que la pondération utilisée est le nombre d'occurrences de termes, la mesure de similarité utilisée est celle d'Euclide. Ainsi, comme prochaine étape, il s'agit de calculer la distance de chaque texte par rapport à chaque centre. A titre d'exemple, la distance euclidienne entre Texte1 et centre  $c_1$  se calcule comme suit.

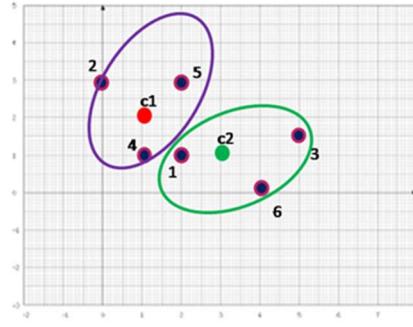
$$d(\text{Texte1}, c_1) = \sqrt{\sum_{j=1}^t (u_i - v_j)^2} = \sqrt{(2-1)^2 + (1-2)^2} = \sqrt{2} = 1.41$$

La figure suivante, montre le résultat de calcul de la matrice de distance et une représentation graphique de la première itération. Nous allons voir comment manipuler ces opérations avec R dans les sections suivantes.

	t1	t2
Texte1	2	1
Texte2	0	3
Texte3	5	2
Texte4	1	1
Texte5	2	3
Texte6	4	0

Matrice de distance			
	C1	C2	cluster
Texte1	1.41	1	2
Texte2	1.41	3.6	1
Texte3	4	2.23	2
Texte4	1	2	1
Texte5	1.41	2.23	1
Texte6	3.6	1.41	2

**c1(1,2) et c2(3,1)**



## Représentation graphique

Remarquons à ce stade que les textes ne sont pas hiérarchisés mais coexistent du fait d'être proche d'un centre par rapport à l'autre. Maintenant, que deux clusters sont construits, il faut recalculer le nouveau centre correspondant à chacun des clusters. Pour ce faire, utiliser la formule (3).

$$\left( \frac{\sum X_i}{Z_i}, \frac{\sum Y_j}{Z_j} \right). \quad \dots \dots \dots \quad (3)$$

L'application de cette formule donne pour résultat un nouveau centre (3.66, 1). Pour l'autre cluster nous obtenons (1, 2.33). Le premier nouveau centre est obtenu comme suit :

$$\left( \frac{2+5+4}{3}, \frac{1+2+0}{3} \right)$$

Une fois les nouveaux centres obtenus, il faut refaire les calculs précédents jusqu'à ce que les barycentres se stabilisent.

## Manipulation de l'exemple avec R.

```
> matrice = as.matrix(data.frame(c(2,0,5,1,2,4), c(1,3,2,1,3,0)))
> k<-kmeans(matrice, centers = rbind(c(1,2),c(3,1)), iter.max = 1,algorithm = c("Lloyd"),
trace=TRUE)
Warning message:
pas de convergence en 1 itération
> k$cluster
[1] 2 1 2 1 1 2
> k$centers
  c.2..0..5..1..2..4. c.1..3..2..1..3..0.
1          1.000000      2.333333
2          3.666667      1.000000
```

Notons que si vous ne spécifiez pas le paramètre par défaut *algorithm*, alors c'est l'algorithme de Hartigan et Wong [19] qui est utilisé par défaut. Il est à noter que certains auteurs utilisent k-means pour faire référence à un algorithme spécifique plutôt qu'à la méthode générale: le plus souvent l'algorithme donné par MacQueen [21] et parfois celui donné par Lloyd et Forgy [18-20]. L'algorithme de Hartigan-Wong fait généralement un meilleur travail que l'un ou l'autre.

## Classification ascendante hiérarchique (HCA)

### Hierarchical cluster analysis

HCA, offre deux approches : la première est dite agglomérative. Cette dernière est connue sous le nom : AGNES (Agglomerative Nesting), considère que chaque texte forme au départ un cluster distinct nommé singleton. Les textes sont fusionnés deux à deux successivement pour grouper deux clusters, et ainsi de suite. Cette opération est répétée jusqu'à ce qu'un critère d'arrêt soit satisfait. Quant à la seconde approche, elle est connue sous le nom de DIANA (DIvisive ANAlysis)). Cette dernière approche considère au départ que tous les textes sont dans un unique groupe (cluster), ensuite effectue la division jusqu'à ce qu'un critère d'arrêt soit satisfait. Contrairement à l'algorithme K-mean, il n'est pas obligatoire de spécifier le nombre de groupes à créer.

De plus, l'approche HCA offre une arborescence des textes connue sous le nom de dendrogramme. La distance de division ou de fusion (appelée hauteur) est indiquée sur l'axe des y du dendrogramme ci-dessous. Ainsi, le principe HCA que nous allons utiliser se résume comme suit :

1. A l'initialisation, chaque individu représente une classe (singleton)
2. Calculer les écarts de similarité/dissimilarité des individus deux à deux selon un critère d'agrégation basé sur les distances (saut minimal, distance maximale ou distance moyenne)
3. A chaque étape est construite une nouvelle partition de deux individus dont la similarité est plus forte.
4. Refaire les étapes 2 et 3.
5. Enfin, l'agrégation de tous les individus en une seule classe.

Les mesures suivantes sont utilisées dans l'algorithme HCA pour le calcul de distance.

- Single linkage (saut ou distance minimal):  $d(c_i, c_j) = \min_{x \in c_i, y \in c_j} \|x - y\|^2$



Il s'agit de prendre la plus petite distance entre éléments de deux clusters. Ne pas oublier qu'un singleton est un cluster en soit.

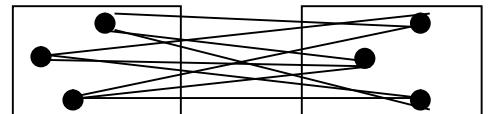
- Complete linkage (distance maximale):  $d(c_i, c_j) = \max_{x \in c_i, y \in c_j} \|x - y\|^2$

Pour grouper deux clusters, on calcul l'indice d'agrégation en prenant en compte plus grande distance entre éléments de deux clusters.



- Average linkage (distance moyenne):  $d(c_i, c_j) = \frac{1}{|c_i||c_j|} \sum_{x \in c_i, y \in c_j} \|x - y\|^2$

Quant à cette méthode de distance, il s'agit de considérer la distance moyenne entre éléments.



### Exemple.

Reprenons l'exemple ci-dessus.

Nous appliquons le CAH avec un critère du saut maximal.

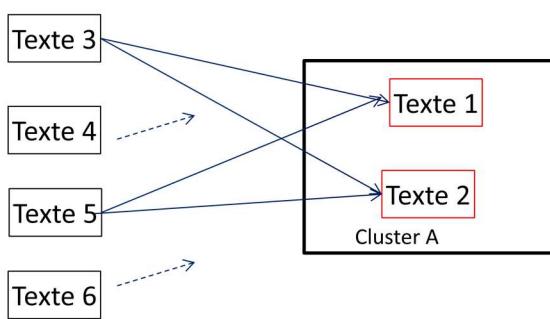
Nous allons d'abord calculer la distance euclidienne entre individus. Le résultat est reproduit par la figure suivante :

	t1	t2	1	2	3	4	5
Texte1	2	1	2.828427				
Texte2	0	3	3.162278	5.099020			
Texte3	5	2	4.100000	2.236068	4.123106		
Texte4	1	1	5.200000	2.000000	3.162278	2.236068	
Texte5	2	3	6.2236068	5.000000	2.236068	3.162278	3.605551
Texte6	4	0					

Étape d'agrégation des individus. Puisqu'il s'agit de la première étape, nous allons agréger les individus les plus similaires en fonction du calcul de distance (critère Euclide), en l'occurrence texte 1 et texte 4 (la valeur de la cellule est la plus petite dans toutes la matrice). Donc notons la nouvelle classe (cluster) A={text1,text4} et notons les autres textes par O2 (texte 2) ,O3 (texte 3), O5 (texte 5) et O6 (texte 6). Nous allons ensuite calculer la distance (saut) entre chacun des textes avec le cluster A. Le calcul de saut se fait comme suit : le cluster A est composé de deux textes, ainsi pour chacun des autres textes, calcul l'écart avec les deux textes text1 et texte4. Un calcul numérique donne ce qui suit :

$$\max (\text{ecart}(\text{text2},\text{text1}), \text{ecart}(\text{text2},\text{text4})) = \max (2.82, 2.23) = \mathbf{2.82}.$$

Nous devons continuer jusqu'à construire la nouvelle matrice de distance basée sur le saut *complete*, figure suivante. Par conséquent, un autre cluster sera agrégé, il s'agit de texte 2 et texte 5 qui sont les plus proches dans la nouvelle matrice. Notons ce nouveau cluster B= {text2,text5}



	O2	O3	O5	O6	A
O2	0	5.09	2	3.60	2.82
O3		0	3.16	2.23	3.16
O5			0	3.60	2
O6				0	3.16
A					0

On calcule une nouvelle fois la matrice de distance avec le critère saut maximal.

	O3	O6	A	B
O3	0	2.23	3.16	5.09
O6		0	3.16	3.60
A			0	2.82
B				0

$$\begin{aligned} \max(O3, B) &= \max(\text{ecart}(O3, O2), \text{ecart}(O3, O5)) = \\ &\max(5.09, 2) = \mathbf{5.09} \\ \max(B, A) &= \max(\text{ecart}(O2, A), \text{ecart}(O5, A)) = \\ &\max(2.82, 2) = \mathbf{2.82} \end{aligned}$$

Le même calcul sera fait sur la matrice, ce qui fait un troisième cluster est généré.  
Ce cluster sera noté C= {texte 3, texte 6}.

On calcule une nouvelle fois la matrice de distance avec le critère saut maximal.

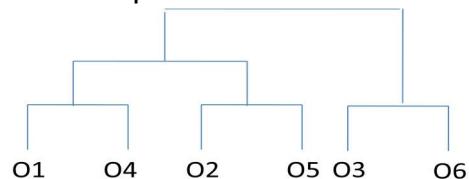
	A	B	C
A	0	2.82	4.12
B		0	5.09
C			0

$$\begin{aligned} C &= \{O3, O6\} \\ \max(A, C) &= \max(\text{ecart}(O1, C), \text{ecart}(O4, C)) = \\ &\max(3.16, 4.12) = \mathbf{4.12} \\ \max(B, C) &= \max(\text{ecart}(O2, C), \text{ecart}(O5, C)) = \\ &\max(5.09, 3.60) = \mathbf{5.09} \end{aligned}$$

On agrège A et B , D={A,B}

On calcule une nouvelle fois la matrice, il s'agit maintenant des deux derniers individus. D et C dont la distance est égale à 5.09.

Enfin, on peut construire le dendrogramme suivant qui retrace la création de l'arbre depuis tout le début.



## Manipulation avec R

Dans ce qui suit, vous tester l'exemple que nous venons de voir ensemble. Il suffit de réécrire les commandes une à une et comprendre les résultats intermédiaires.

```
15
16 o1=c(2,0,5,1,2,4)
17 o2=c(1,3,2,1,3,0)
18 m<-cbind(o1,o2)
19 m
20 matde<-dist(m, method = "euclidean")
21 matde
22 hcTree<-hclust(matde, "complete")
23 hcTree$height
24 plot(hcTree, cex=0.7,hang=-1)
25 hcTree$merge
```

18:1 (Top Level) R Script

Console Terminal ×

~/dataset/ ↗

```
> o2=c(1,3,2,1,3,0)
> m<-cbind(o1,o2)
> m
   o1 o2
[1,] 2 1
[2,] 0 3
[3,] 5 2
[4,] 1 1
[5,] 2 3
[6,] 4 0
> matde<-dist(m, method = "euclidean")
> matde
      1         2         3         4         5
2 2.828427
3 3.162278 5.099020
4 1.000000 2.236068 4.123106
5 2.000000 2.000000 3.162278 2.236068
6 2.236068 5.000000 2.236068 3.162278 3.605551
> hcTree<-hclust(matde, "complete")
> hcTree$height
[1] 1.000000 2.000000 2.236068 2.828427 5.099020
> hcTree$merge
[,1] [,2]
[1,] -1   -4
[2,] -2   -5
[3,] -3   -6
[4,]  1    2
[5,]  3    4
```

Matrice de distance

Les valeurs d' inerties correspondant à chacune des matrices de distance

Correspond aux pas d'agrégation Vérifier ci-dessus les cellules de chaque matrice de distance lors d'un nouveau calcul

## Exercice.

Essayer d'exécuter manuellement en utilisant le saut minimum, ensuite le saut avec la distance moyenne et vérifier avec R vos résultats.

## Critère de Ward

Dans diverses applications, le saut minimum peut créer un effet de chaîne qui permet de regrouper des entités de proche en proche. Ce problème ne permet pas de mettre en évidence la séparation entre deux classes. L'utilisation du critère de Ward comme un critère de distance permet d'y remédier à ce phénomène.

Avec *ward*, il y a prise en compte de la dispersion dans un groupe et entre groupes. (i.e. minimiser l'inertie intra-classe est équivalent à maximiser l'inertie inter classes).

1. Comme de CAH (HCA : hierarchical cluster analysis). Nbr singleton=Nbr document
2. Ce qui change est le calcul de la matrice de données associée.

$$d^2(A, B) = \frac{\mu_A \mu_B}{\mu_A + \mu_B} d^2(g_A, g_B) \quad \dots \dots \dots \text{(ward)}$$

$g_A$ : centre de gravité du cluster,

$\mu_A$ : nombre d'individu dans le groupe

### Exemple.

Considérons l'exemple ci-dessous pour dérouler le HCA avec le critère de Ward.

	t1	t2
text1	3	6
text2	2	0
text3	1	2
text4	2	7
text5	0	4

Au départ, il y a cinq singltons (clusters) {text1 text2 text3 text4 text5}. Ainsi, calculer la distance entre text1 et text2 avec le critère de ward est comme suit :

$\mu_A = 1$  et  $\mu_B = 1$  (voir formule de ward). Donc l'écart associé à ces deux textes.  
(Chaque singleton forme son propre centre de gravité).

$$d(\text{text1}, \text{text2}) = \frac{1+1}{1+1} d^2((3-2) + (6-0)) = \frac{1}{2} ((1) + (36)) = \frac{37}{2} = 18.5$$

	text1	text2	text3	text4	text5
text1	0				
text2	18.5	0			
text3	10	2.5	0		
text4	1	24.5	13	0	
text5	6.5	10	2.5	6.5	0

A partir de la matrice, le résultat du regroupement permettant de minimiser l'inertie interclasse se traduit par le premier couple  $c1=\{\text{text1}, \text{text4}\}$ . Ces deux derniers sont les plus proches.

Comme dans l'exemple HCA sans le critère de ward, nous avons une nouvelle partition composée du nouveau couple et les autres textes ( $\text{text2}$ ,  $\text{text3}$ ,  $\text{text5}$ ,  $c1$ ). Ainsi, nous allons calculer l'inertie intra classe.

$$\text{Intra}(\Gamma_0) = \text{Intra}(\text{text2}, \text{text3}, \text{text5}, c1) = 1/5 * 1 = \mathbf{0.2}$$

Avant de continuer, il faudra calculer le centre de gravité de  $c1$ .

$$\mathbf{gc1} = ((2+3)/2, (6+7)/2) = (2.5, 6.5)$$

text1	3	6
text4	2	7

Recalculons la distance de chaque point par rapport à ce nouveau centre.

$$d(\text{text2}, \mathbf{gc1}) = \frac{1+2}{1+2} d^2((2-2.5) + (0-6.5)) = \frac{2}{3} d^2((0.25) + ((42.25))) = \frac{85}{3} = 28.33$$

/  $\mu_{c1} = 2$

text2	2	0
text3	1	2
text5	0	4

Après avoir effectué tous les calculs (si pas d'erreur de calcul) on obtient la nouvelle matrice de distances suivante :

		text2	text3		text5	C1
text2		0				
text3		2.5	0			
text5		10	2.5		0	
C1		<b>28.33</b>	<b>15</b>		<b>8.33</b>	0

Un nouveau cluster nommé C2 peut être formé, il s'agit de text2, text3.

$$\text{Intra}(\Gamma_1) = \text{Intra}(\text{text5}, \text{C1}, \text{C2}) = 0.2 + (1/5 \times 2.5) = \mathbf{0.7}$$

Calculer le centre de gravité du nouveau cluster C2

text3	1	2
text2	2	0

$$\mathbf{gc_2} = ((1+2)/2, (2+0)/2) = (1.5, 1) / \text{C2} = \text{texte2} \cup \text{texte3}.$$

Recalculons la distance associée au nouveau centre.

	texte5	C1	C2
texte5	0		
C1	8.33	0	
C2	<b>7.5</b>	31.25	0

text5	0	4
<b>gc<sub>1</sub> = (2.5, 6.5)</b>		
<b>gc<sub>2</sub> = (1.5, 1)</b>		

Avec  $d(C1, C2) = \frac{\sqrt{2^2 + 2^2}}{2+2} = \sqrt{(2.5 - 1.5)^2 + (6.5 - 1)^2} = \sqrt{4} = 2$

$$(5.5)) = \frac{4(1+30.25)}{4} = 31.25$$

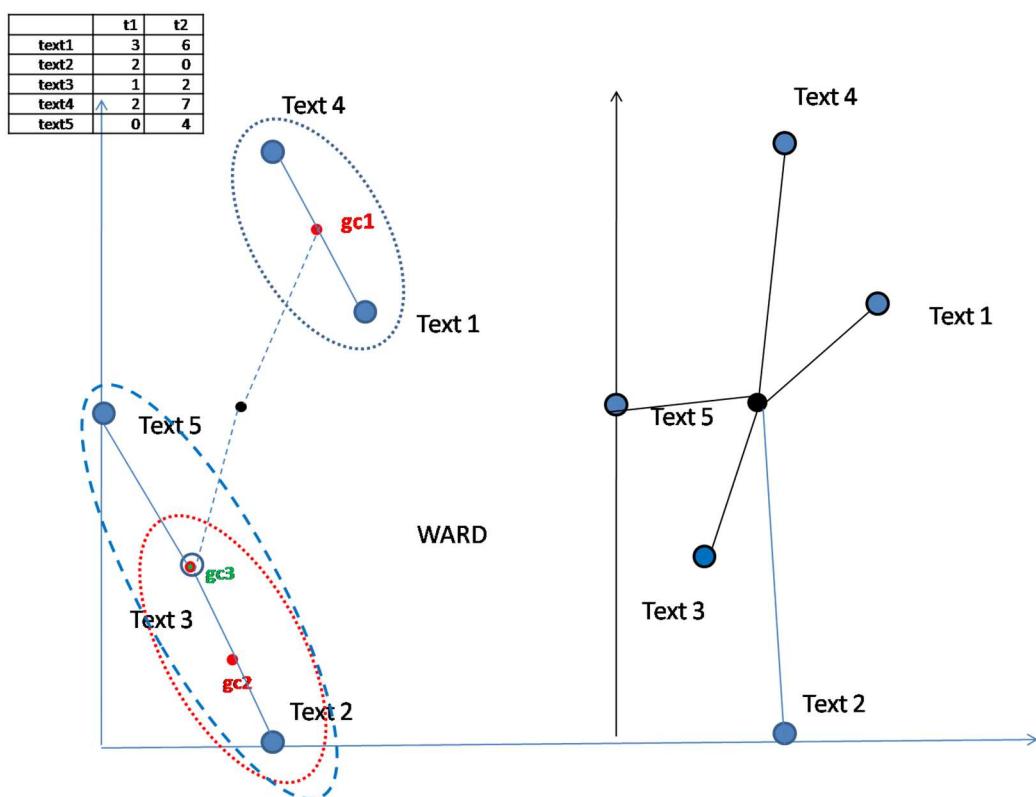
Le nouveau couple qui se forme est C3=texte5 U C2. Donc  $\mathbf{gc_3} = (1, 2)$ .

$$\text{Intra}(\Gamma_2) = \text{Intra}(C3, C1) = 0.7 + (1/5 \times 7.5) = \mathbf{2.2}$$

	C3	C1
C3	0	
C1	27	0

inertie total :  $2.2 + (1/5 \cdot 27) = 7.6$ .

Il devient évident que minimiser l'inertie intra-classe est équivalent à maximiser l'inertie inter-classe. Dans notre exemple, ceci peut être vérifié avec la décomposition de HUGEN<sup>11</sup>, schéma suivant.



Le théorème de Huygens s'appuie sur la décomposition de l'inertie, côté gauche de la figure ci-dessus. Remarquons que les 5 distances (à droite de la figure) au carré entre les textes et leur barycentre sont égales aux 7 distances représentées à gauche de la figure. Formellement cela se traduit par :

---

<sup>11</sup> Mathématicien, astronome et un physicien néerlandais, né le 14 avril 1629 à La Haye et mort le 8 juillet 1695 dans la même ville.

$$x_1 = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_{1,i} - \bar{x}_1)^2} \quad x_2 = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_{2,i} - \bar{x}_2)^2}$$

De ce qui précède, nous avons  $x_1 = (1.019)^2$  et  $x_2 = (2.56)^2 = 7.59$ . ce qui confirme le résultat obtenu avec la décomposition de ward à savoir  $0.2 + 0.5 + 1.5 + 5.4 \approx 7.59$

## Manipulation de textes avec R

Je suppose dans ce qui suit, que la variable mydocs est utilisée pour désigner le corpus. Attention, il faut d'abord être sûr que votre répertoire contient uniquement les fichiers de travail, et n'oublier pas de charger la librairie *tm* si ce n'est pas encore fait, voir figure suivante.

```
setwd("G:\\M1 tic 2020\\Cours Off 2020\\scripts\\exemplecours\\docs2")
getwd()
mydocs<-Corpus(DirSource(getwd()))
```

Afin de vérifier le contenu de votre répertoire, deux méthodes se présentent à vous. Soit, vous visualiser sur la console le contenu d'un seul fichier à la fois, soit, visualiser le contenu de tous les fichiers en même temps en utilisant la commande **inspect(mydocs)**. Si vous voulez visualiser le 3<sup>ème</sup> fichier, il suffit d'exécuter `inspect(mydocs[3])` ou `writeLines(as.character(mydocs[1]))`. Ces commandes sont utiles pour suivre l'évolution d'un document.

```
> inspect(mydocs[1])
<<SimpleCorpus>>
Metadata: corpus specific: 1, document level (indexed): 0
Content: documents: 1

a good health.txt
people who exercise live longer and keep well threei health on average than those
who don't. \nDozens of studies show that regular physical activity lowers your
risk of heart disease, stroke, diabetes, \n and avoid other health problems.
```

## Prétraitement des textes

En vue de préparer les textes et de créer la matrice terme documents. Nous devons faire appel à l'interface `tm_map`, une interface pour appliquer les fonctions de transformations avec le langage R. Pour plus de détails faire `help(tm_map)`. Pour supprimer les apostrophes et autres caractères, la fonction « `content_transformer` » est utilisée. Dans le help de R, vous pouvez vérifier que cette fonction modifie le contenu d'un objet R. Son usage consiste à passer en paramètre une fonction dédiée à une transformation donnée. Je vous conseille de visualiser le contenu d'au moins un fichier pour voir l'effet des commandes.

### Transformer les lettres (caractères) en minuscules

```
mydocs <- tm_map(mydocs, content_transformer(tolower))  
mydocs <- tm_map(mydocs, removeNumbers)
```

La fonction `toSpace` ci-dessous utilise `content-transformer` pour modifier la valeur indiqué dans la variable `pattern` en un espace (blanc). La valeur espace est indiquée comme second paramètre dans le résultat retourné par la fonction `content_transformer` :

```
toSpace <- content_transformer(function(x, pattern) { return (gsub(pattern, " ", x))})
```

Pour appliquer cette fonction au corpus `mydocs`, il suffit d'appliquer la fonction `toSpace` via `tm_map` comme suit :

```
mydocs <- tm_map(mydocs, toSpace, "") # supprime apostrophe
```

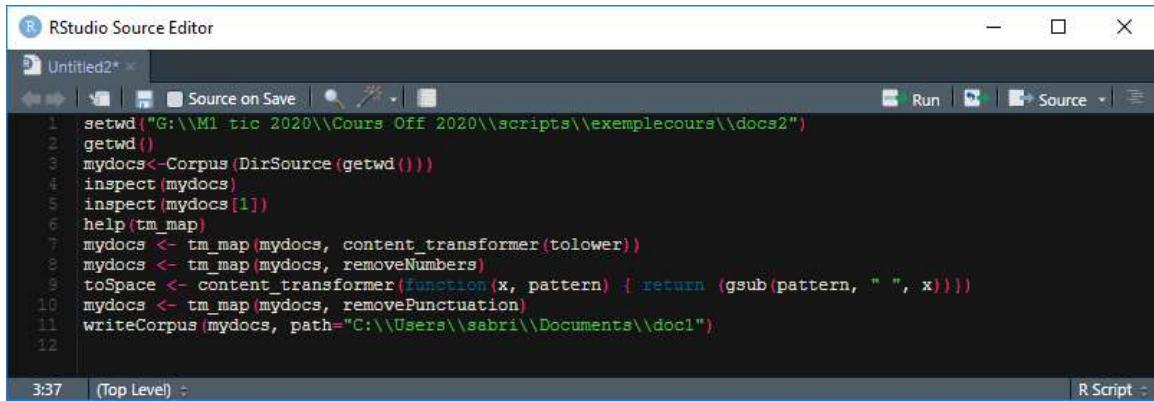
Si vous avez mots qui contiennent des trait d'union (-), vous pouvez utiliser la commande suivante.

```
mydocs<-tm_map(mydocs,toSpace,"-") # supprime le trait d'union  
mydocs <- tm_map(mydocs, removePunctuation)
```

Avec la commande suivante, vous allez pouvoir copier le contenu des fichiers traités jusqu'ici. Toutefois, il faut créer dans un premier temps un répertoire nommé par exemple **doc1** dans le répertoire de votre choix. Il est impératif de mettre double slash (barre renversé) si vous êtes sous Windows.

```
writeCorpus(mydocs, path="C:\\Users\\sabri\\Documents\\doc1")
```

Vérifier que le répertoire doc1, contient le résultat des fichiers ayant subis les traitements. Ouvrir un fichier pour vérifier. Cette façon de faire est très utile pour revenir faire des tests ultérieurement.



The screenshot shows the RStudio Source Editor window. The title bar says "RStudio Source Editor". The main area contains the following R code:

```
1 setwd("G:\\M1_tic_2020\\Cours Off 2020\\scripts\\exemplécours\\docs2")
2 getwd()
3 mydocs<-Corpus(DirSource(getwd()))
4 inspect(mydocs)
5 inspect(mydocs[1])
6 help(tm_map)
7 mydocs <- tm_map(mydocs, content_transformer(tolower))
8 mydocs <- tm_map(mydocs, removeNumbers)
9 toSpace <- content_transformer(function(x, pattern) { return (gsub(pattern, " ", x))})
10 mydocs <- tm_map(mydocs, removePunctuation)
11 writeCorpus(mydocs, path="C:\\Users\\sabri\\Documents\\doc1")
12
```

The status bar at the bottom left shows "3:37" and "Top Level". The status bar at the bottom right shows "R Script".

## Suppression des mots vides

Pour manipuler des textes avec le langage R, il existe plusieurs outils, dans notre cas, nous allons manipuler les librairies *textstem*, *korporus*, des outils tels que *dplyr* et *tidytext* ont pour rôle de faciliter l'écriture du code. Ainsi, nous allons manipuler un peu des fonctionnalités de chacun de ces outils. L'étudiant est encouragé à enrichir ces connaissances à partir des sites dédiés à ce sujet.

```
mydocs <- tm_map(mydocs, removeWords, stopwords("fr"))
```

Pour supprimer les mots vides anglais utiliser la commande suivante :

```
mydocs <- tm_map(mydocs, removeWords, stopwords("english"))
```

Pour connaître la liste des mots vides Français utilisés dans R, exécuter la commande suivante : `stopwords(c("fr"))`.

Si vous vérifiez le contenu des fichiers, vous allez sûrement trouver qu'il en reste encore des mots que l'on peut considérer comme inutiles (mots vides) et des espaces

en trop qu'il va falloir supprimer. Pour ce faire, vous pouvez créer une liste de ces mots et les supprimer. De mon côté, j'ai créé par exemple la liste suivante :

```
maListeDeMotsASupprimer <-  
c("around", "guide", "caused", "widespread", "modern", "different", "parts", "diverse", "high"  
, "workrelated", "well", "certain", "prolonged", "sophisticated", "advanced", "profitable",  
"efficient", "vary", "depending", "part", "body", "depend", "understand", "output", "strength"  
, "always", "main", "followed", "principal", "many")  
  
mydocs <- tm_map(mydocs, removeWords, maListeDeMotsASupprimer)
```

Dans la suite de ce travail, je considère que les mots suivants peuvent être supprimés du fait qu'ils n'apportent plus aucun sens et que à priori le contexte n'est pas pris en compte dans notre analyse. Quant à la suppression des espaces en trop générés lors du prétraitement des textes, utiliser cette commande:

```
mydocs <- tm_map(mydocs, stripWhitespace)
```

Ne pas oublier de recopier les fichiers pour copier le résultat dans doc1.

## Effectuer la lemmatisation et la racinisation

Ne pas oublier de charger les librairies : library(tm) library(textstem)  
library(koRpus) library(koRpus.lang.en) library(syll) library(SnowballC).

Ensuite afficher sur la console le fichier 7. Dans ce fichier il existe le terme « *used* ».

Ensuite exécuter la commande suivante :

```
mydocs <- tm_map (mydocs, lemmatize_strings)
```

Remarquer que suite à cette opération le verbe *used* a été lemmatisé (forme infinitif). Comme vous pouvez remarquer que des mots sont transformés du pluriel au singulier. Utiliser la fonction *inspect* sur l'ensemble du corpus pour visualiser le contenu de vos fichiers. Vous avez dû sûrement remarquer que d'autres mots peuvent

être encore supprimés. Enfin, il reste une dernière opération à exécuter, il s'agit du stemming.

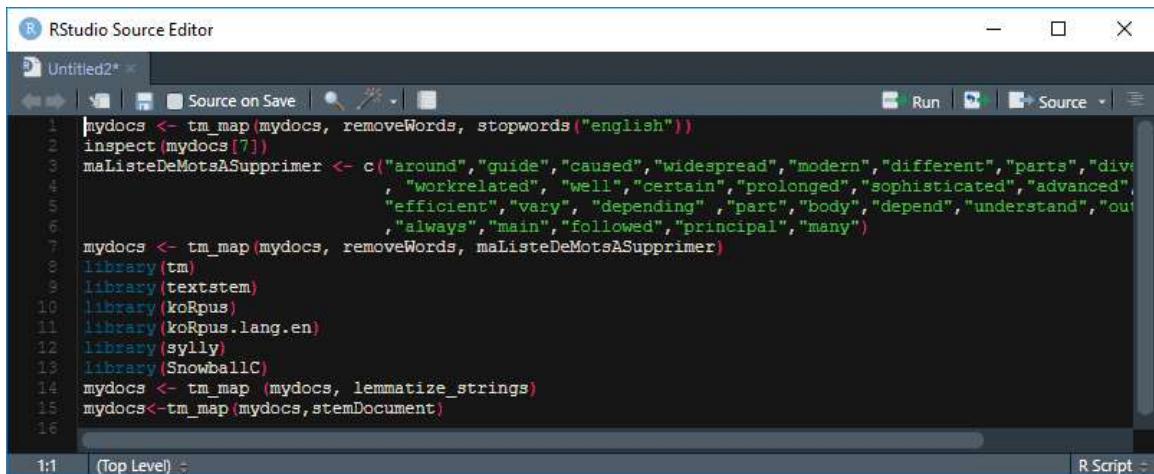
```
mydocs <- tm_map(mydocs, stemDocument, language="french")
```

La langue peut être omise quand il s'agit de l'Anglais :

```
mydocs<-tm_map(mydocs,stemDocument) ou cette commande
```

```
mydocs <- tm_map (mydocs, stem_strings).
```

Attention, il est parfois nécessaire d'exécuter le stemming plusieurs fois pour obtenir le bon résultat.



The screenshot shows the RStudio Source Editor window. The title bar says "R RStudio Source Editor". The tab bar has "Untitled2\*" selected. The code editor contains the following R script:

```
1 mydocs <- tm_map(mydocs, removeWords, stopwords("english"))
2 inspect(mydocs[7])
3 maListeDeMotsASupprimer <- c("around", "guide", "caused", "widespread", "modern", "different", "parts", "diverse",
4                                "workrelated", "well", "certain", "prolonged", "sophisticated", "advanced",
5                                "efficient", "vary", "depending", "part", "body", "depend", "understand", "out",
6                                "always", "main", "followed", "principal", "many")
7 mydocs <- tm_map(mydocs, removeWords, maListeDeMotsASupprimer)
8 library(tm)
9 library(textstem)
10 library(koRpus)
11 library(koRpus.lang.en)
12 library(syll)
13 library(SnowballC)
14 mydocs <- tm_map (mydocs, lemmatize_strings)
15 mydocs<-tm_map(mydocs,stemDocument)
```

Pour le texte devices.txt (numéro 7), après un prétraitement, j'ai obtenu le résultat suivant.

```
devices.txt
robot sensor use estim robot condit environ signal pass control enabl appropri
behavior
```

Une fois que les textes sont traités, on peut maintenant commencer à appliquer des algorithmes de clustering ou de classification. Pour se faire, il faut suivre les étapes suivantes :

### Création de la matrice : Dtm=DocumentTermMatrix

Cette matrice va permettre de représenter les données sous forme de vecteurs.

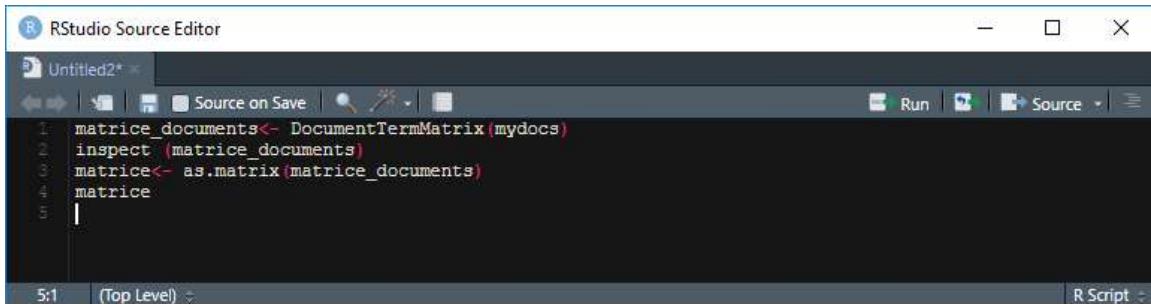
Exécutez la commande :

```
matrice_documents<- DocumentTermMatrix(mydocs)
```

Sélectionner ensuite la variable *matrice\_documents* et faire un **run**, quelques informations (statistiques) utiles seront affichées sur la console. Vous pouvez également faire `inspect(matrice_documents)`.

Pour visualiser l'ensemble des éléments constituant la matrice de documents, exécuter successivement les commandes suivantes et faire un clic sur la grille à côté du nom de la matrice dans l'onglet Environment de RStudio.

- `matrice<- as.matrix(matrice_documents)`
- `matrice`



Comme vous avez pu le constater, les textes étudiés ne sont pas volumineux étant donné que l'objectif est d'avoir une prise en main et comprendre les principes exposés tout au long de ce document. Par conséquent, dans notre cas, la fréquence d'un mot n'a aucun sens. Par contre, pour des documents volumineux, il serait utile de supprimer les mots dont la fréquence est moins de 20 % par exemple. Dans ce dernier cas, il suffit d'exécuter la commande : `removeSparseTerms(matrice_documents, Taux_voulu))# Taux_voulu= 0.4` (signifie 40%)

Exédez les commandes suivantes :

- `freq <- colSums(as.matrix(matrice_documents))`
- `freq`
- `length(freq) : Renvoie le nombre de mots dans la matrice`
- `findFreqTerms(matrice_documents, lowfreq = 4) : Renvoie les mots qui apparaissent au moins quatre fois dans le corpus.`

Vous pouvez modifier le chiffre pour trouver ceux qui apparaissent 1,2, 4 fois. etc.

freTerme<-findFreqTerms(matrice\_documents, lowfreq = 2) est une commande pour retenir les termes apparaissant au moins deux fois.

Pour obtenir les mots fréquents, il suffit d'afficher la queue ou la tête d'un vecteur. Grâce à la commande *order(freq,decreasing=TRUE)* la fréquence dans ordre décroissant de termes dans le corpus est renvoyé, il devient ainsi facile d'obtenir les termes les plus fréquents et les moins fréquent en utilisant respectivement les deux dernières commandes suivantes:

```
rdr <- order(freq,decreasing=TRUE)  
freq[head(ordr)]  
freq[tail(ordr)]
```

## Trouvez les associations entre les mots

Il s'agit ici de découvrir la corrélation des mots avec un terme donné. Pour ce faire, la fonction *findAssocs* de la librairie tm est utilisée. Attention il faut également fournir une valeur comme limite de corrélation en argument. La figure suivante donne un aperçu de ce que vous devriez obtenir.

```
findAssocs(matrice_documents, terms = "sensor", corlimit = 0.5)
```

```
> findAssocs(matrice_documents, terms = "sensor", corlimit = 0.4)  
$sensor  
    can   monitor   robot  appropri  action   analyz   final   imag  intermedi   reach  
0.67     0.64     0.49     0.48     0.45     0.45     0.45     0.45     0.45     0.45  
surround  visual  analog  explicit lack.phenomenon  provid  communiti  concept  creat  
0.45     0.45     0.45     0.45     0.45     0.45     0.45     0.45     0.45     0.45  
event    fuse    happen  internet iort    togeth movement seetouch  datum  
0.45     0.45     0.45     0.45     0.45     0.45     0.44     0.44     0.44
```

Il est possible d'avoir un visuel graphique des associations entre termes. Ce moyen de le faire est facilité par des librairies que vous devez devoir installer (si des mises à jour sont nécessaires, il faut accepter): *install.packages("BiocManager")*, ensuite exécuter la commande *BiocManager::install("Rgraphviz")*. Charger les librairies :

*library(graph)* *library(Rgraphviz)*, et enfin, exécuter les commandes suivantes :

```
freTerme<-findFreqTerms(matrice_documents, lowfreq = 5)  
plot(matrice_documents, term = freTerme, corThreshold = 0.5)
```

Pour avoir un aperçu graphique des mots dont la taille est proportionnelle à la fréquence d'un mot dans le corpus, nous allons utiliser les commandes suivantes pour afficher un nuage de mots (wordcloud) . Pour ce faire, Installer le package wordcloud. Ensuite charger ce package library(wordcloud).

- ```
> library(wordcloud)
> wordcloud(mydocs,min.freq=2,random.order=FALSE,colors=
  brewer.pal(3,"Dark2"))
```

**NOTE :** La méthode worldcloud peut avoir d'autres paramètres comme option. Pour plus d'informations, je vous invite à lire le pdf en ligne

<https://cran.r-project.org/web/packages/wordcloud/wordcloud.pdf>



Modifier min.freq pour voir l'effet que cela puisse avoir.

**Remarque :** Il se peut que vous n'ayez pas exactement le même résultat si vous avez omis ou bien supprimer beaucoup plus de mots que vous considérez comme vide.

## Exercice

Dans notre travail, nous avons supprimé des mots vides. Il vous est demandé de créer le nuage des mots sans supprimer ces mots vides.

## **Conception d'un histogramme :**

En statistique, un histogramme a pour objectif de mesurer la distribution de valeurs d'une variable. Pour le texte mining, cette étape peut s'avérer utile. Toutefois cette représentation ne peut pas être une représentation objective des données (il faut concevoir un graphe pouvant représenter une sémantique des données tel que RDF(S)). Pour manipuler un ensemble de donnée en vue de dessiner un histogramme des mots à partir des textes dans le langage R, il est plus facile de passer par un type de data.frame. Ce type de structure peut être considéré comme une matrice. Pour se faire, dans un premier temps, installer les librairies ggplot2 et tidytext, et ne pas oublier de les charger, dans un second temps, utilisée la commande suivante pour transformer la matrice des mots fréquents en un data.frame :

```
freq <- colSums(as.matrix(matrice_documents))

mydata=data.frame(terme=names(freq), nombre_occurrences=freq)

mydata # faire Run
```

Comme vous pouvez faire un *print(mydata)*, ou visualiser à partir de l'onglet environment.

Afin de créer d'avoir un graphe des mots fréquents, nous allons exécuter les instructions ci-dessous dans l'ordre. Je préfère éclater la fonction ggplot pour une meilleure compréhension.

```
p<-ggplot(subset(mydata, freqr>2), aes(terme, nombre_occurrences))
```

Explication : cette commande cherche à créer un sous ensemble de données des mots apparaissant au moins une fois (le choix de 2 ici est justifié par le fait que nos textes ne sont pas volumineux), sinon on aurait pu choisir 10, 20, 50, ... selon le domaine et l'objectif cible. Tout de même, essayer avec la valeur 0, ensuite 1 pour voir ce que cela donne. La fonction aes (aesthetics :esthétique) quant à elle permet d'annoter les axes abscisses et ordonnées.

```
p <- p + geom_bar(stat="identity")
```

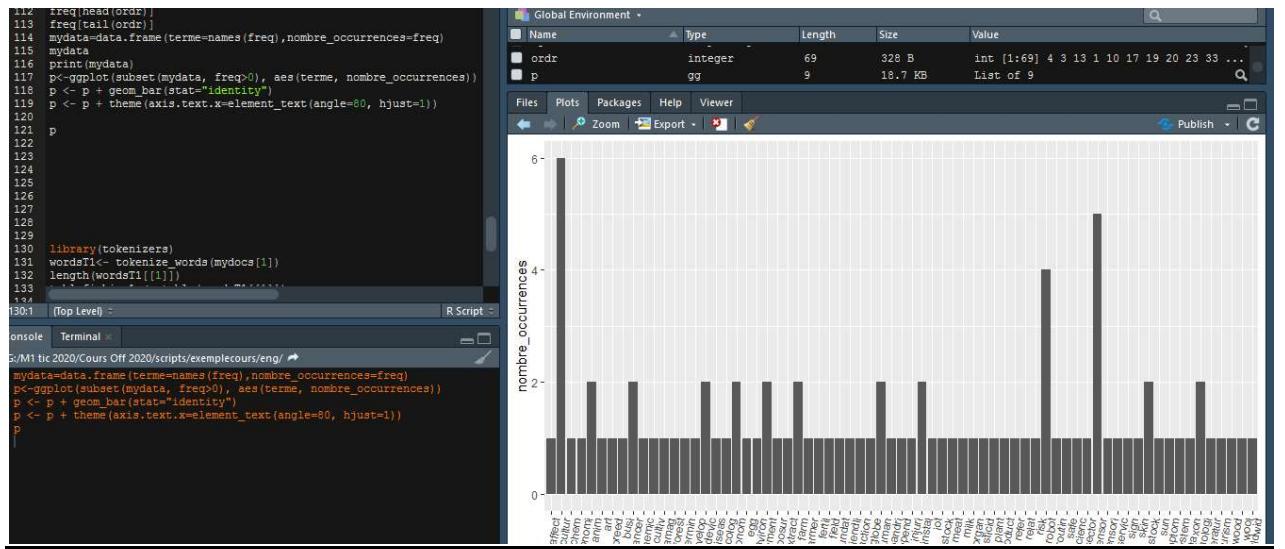
Explication : cette fonction permet de donner une forme de barre ou un nuage de points si la fonction geom\_point n'est pas utilisée.

```
p <- p + theme(axis.text.x=element_text(angle=80, hjust=1))
```

Explication : Les thèmes sont un moyen puissant de personnaliser les composants non liés aux données des tracés: titres, étiquettes, polices, arrière-plan, quadrillage et légendes. Le paramètre hjust : justification horizontale. Pour plus de détails n'hésitez pas à faire un help dans R.

```
p # dernière instruction pour afficher l'histogramme de vos données.
```

Vous devriez avoir un résultat comme celui illustré par la figure suivante.



Vous devez créer dans un premier temps un répertoire dans *Documents* par exemple (vous pouvez choisir un autre emplacement). Dans ce qui je suppose que le répertoire est nommé **docsSerialiser**.

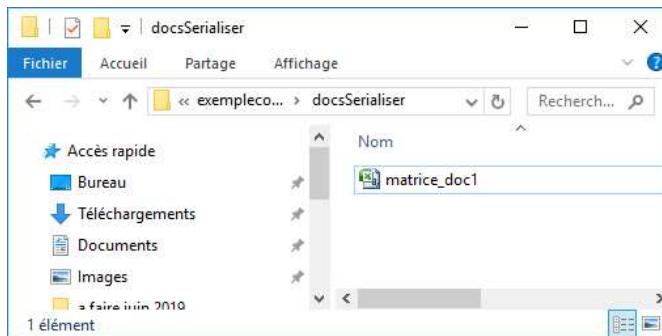
1. Copier tous les fichiers textes de travail dans un répertoire nommé dataset
2. Exécuter la commande suivante : `setwd("path\\ dataset")` . Le path est le chemin correspondant où vous avez créé le répertoire dataset. Dans mon cas, le path est : **C:\\Users\\SABRI\\Documents\\**
3. Vous allez créer un fichier csv que vous allez sérialisez dans le répertoire *docsSerialiser*. Pour ce faire, exécuter la commande suivante :
4. `write.csv2(matrice,file="path\\matrice_doc1.csv",row.names=T)`.

Je rappelle que la variable matrice est créée à partir de la matrice terme documents :

```
matrice<- as.matrix(matrice_documents)
```

➤ Attention :path est le chemin pour atteindre le répertoire *docsSerialiser*, quant à matrice\_doc1.csv est le nom du fichier qui correspond à la matrice de données.

5. Aller dans le répertoire *docsSerialiser*, vous devriez trouver un fichier comme illustrer par la figure suivante



6. Ouvrir le fichier matrice\_doc1 avec votre un éditeur de textes. Notons au passage que le séparateur utilisé est le ; (si vous ouvrez le fichier avec Notepad par exemple).
7. Pour réutiliser les données dans une autre application, Il va falloir utiliser la commande suivante pour charger le fichier dans une matrice (attention parfois

une boîte de dialogue en arrière-plan s'affiche, il va falloir parcourir le PC pour charger le fichier de données sérialisé dans matrice\_doc1.csv).

- my\_old\_matrice <-  
read.csv2(file.choose(),header=TRUE,sep=";",row.names=1)
- my\_old\_matrice # afficher le contenu de la matrice, et , vérifier que les données ont été bel et bien sérialisées.

Une fois vous avez fini de vérifier l'exactitude des données. Vous pouvez continuer votre travail.

### Exercice

Il vous est demandé (vous serrez amener à donner des explications pendant la séance de TD) d'essayer en fonction de votre interprétation, de créer 2 clusters (classes), ensuite, dans chacune des deux classes créées, mettre les textes qui vous semblent le plus similaires possibles en termes de contexte, thème, etc. Ensuite créer trois classes. Ainsi, l'objectif est de trouver des relations entre les textes (observations dans le sens Data Mining) permettant d'identifier les textes similaires (qui se ressemblent le plus) pour les classer dans une même classe. Dans le cas d'absence de variable d'entraînement, nous parlons alors d'une approche non supervisée.

## Catégorisation avec K-means

Il faut utiliser la matrice termes documents. Nous allons utiliser la distance Euclide pour calculer la similarité entre textes. Pour ce faire, la première commande permet de calculer la distance entre les documents. Afficher le résultat qui se trouve dans la variable.

```
distance_matrice<-dist(matrice, method = "euclidean")
```

Utiliser l'algorithme K-means en précisant le nombre de cluster souhaités. Cela se traduit par le paramètre *centers*. Les centres peuvent être explicités ou laisser par défaut. Enfin, un autre paramètre utile est *nstart*. Ce dernier tente plusieurs configurations initiales et rend compte de la meilleure. La valeur recommandée est 25

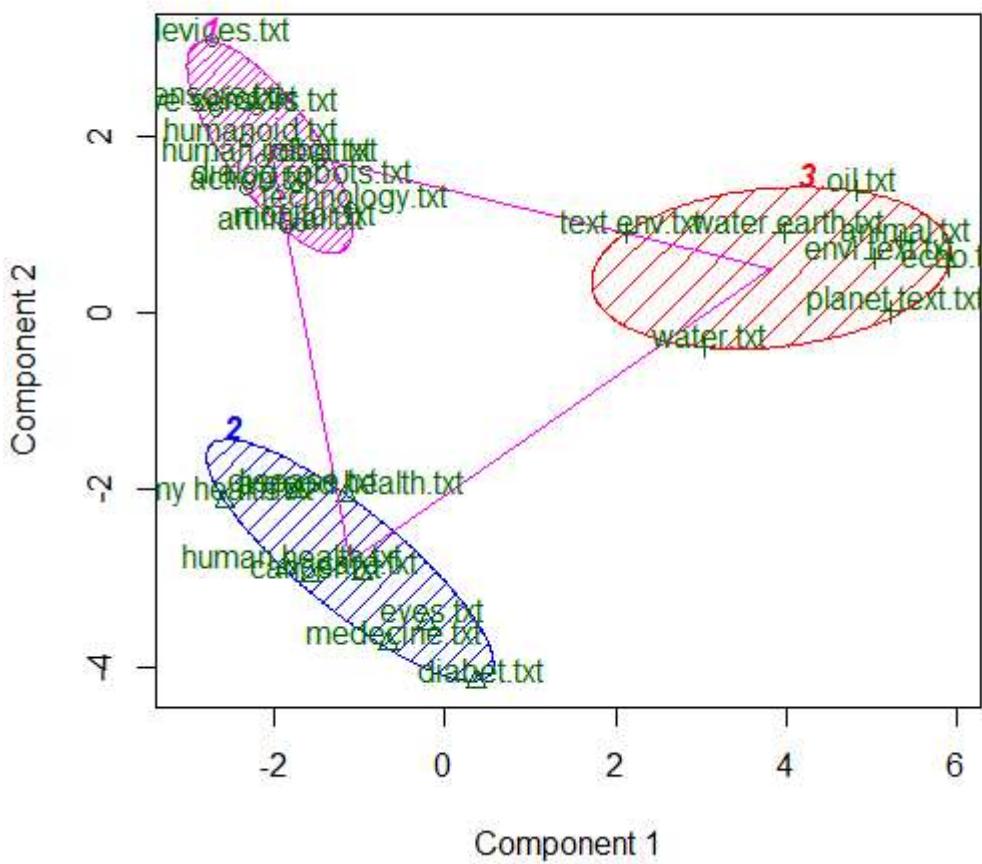
(lire le help dans R). Ainsi, les commandes suivantes doivent être exécutées (vous trouverez l'explication juste après):

- ca=kmeans(nomdevotrematrice,3,nstart=25)
- str(ca)
- ca\$cluster # Cette commande renvoie les clusters et les textes assignées à chacun d'eux.
- ca\$size : indique le nombre d'éléments dans chaque cluster
- round(ca\$centers, digits = 2) vous renvoie les centres utilisés sur deux chiffres après la virgule.

Les paramètres de la commande ci-dessus doivent être interprétés comme suit : 3 représente le nombre de classes (dans le jargon de K-means, on parle de centre). La seconde commande, permet quant à elle d'afficher les noms des fonctions que vous pouvez utiliser sur la variable *ca*, et, donne également un résumé des opérations exécutées par Kmeans. La troisième commande, permet donc d'utiliser un des composants renvoyés par l'algorithme Kmeans tels que *cluster* pour afficher l'assignation (répartition) de chaque texte à une des trois classes. Vous pouvez essayer les autres composants. Il faut remarquer que les centres sont aléatoires. Je vous invite à comparer le résultat affiché avec *ca\$cluster*, avec ce que vous avez proposé. La figure suivante illustre ce résultat. Pour obtenir un graphe de la répartition de vos textes, il suffit d'exécuter les deux commandes suivantes :

- library(cluster)
- clusplot((as.matrix(nomdevotrematrice)), ca\$cluster, color=T, shade=T, labels=2, lines=1)

## CLUSPLOT( as.matrix(distance\_matrice) )



### Comment déterminer le nombre de clusters minimal

Pour un corpus très grand (des milliers de textes) le choix du nombre de clusters s'avère très difficile. En effet, si on choisit un nombre de classes réduit, cela aura comme conséquence des textes non forcément homogènes, donc trop généraliste. Par contre, un très grand nombre de classes (clusters) peut disperser les informations inutilement empêchant donc de déduire des patterns intéressants dans ces textes. La question est comment choisir  $k$  en sorte que la partition soit optimale. Une des solutions possibles consiste à lancer K-means plusieurs fois et observer les résultats

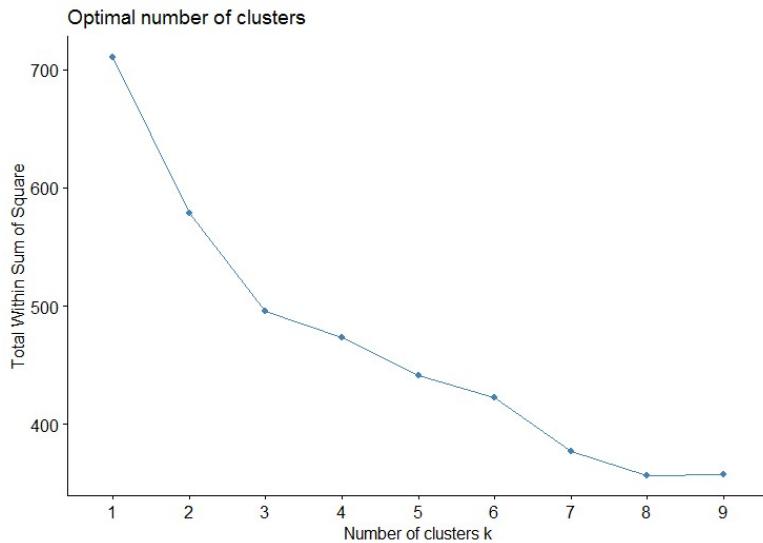
pour en déduire le meilleur k cluster. Il s'agit ici de calculer la variance<sup>12</sup> dans les différents clusters. Une meilleure partition est celle qui maximise les distances inter classes et minimise donc les distances intra-classes. Pour se faire, plusieurs propositions existent. La plus connue est la méthode *Elbow*.

Avec R, il est utile d'utiliser la librairie factoextra<sup>13</sup>. Elle fournit des fonctions faciles à utiliser pour extraire et visualiser la sortie d'analyses de données multivariées, y compris 'PCA' (analyse en composantes principales), 'CA' (analyse de correspondance), 'MCA' (analyse de correspondance multiple), 'FAMD' ( Analyse factorielle des données mixtes), fonctions «MFA» (analyse factorielle multiple) et «HMFA» (analyse factorielle multiple hiérarchique) de différents packages R. Cette librairie contient également des fonctions pour simplifier certaines étapes d'analyse de clustering et fournit une visualisation élégante des données basée sur «ggplot2».

Comme le montre la figure suivante, à partir de k=4, la variation ne se réduit pas de manière significative. Toutefois, la valeur k=6 semble une piste ??

### Exercice :

Créer 6 clusters avec K-means et vérifier manuellement si c'est adequat ?




---

<sup>12</sup> L'ensemble des distances entre le barycentre et les éléments constituant le même cluster

<sup>13</sup> <https://cran.r-project.org/web/packages/factoextra/index.html>

Pour obtenir le graphe ci-dessus il faut installer le package factoextra<sup>14</sup> si ce n'est pas encore fait. Ensuite, charger-la, et enfin exécuter la commande suivante.

```
fviz_nbclust(matrice, kmeans,k.max=9, method = "wss")
fviz_nbclust(matrice, kmeans,k.max=8, method = "gap_stat")
```

## Catégorisation avec HCA

Exécuter les commandes suivantes. Chaque commande sera suivie d'un commentaire pour introduire une explication. D'autres commandes peuvent être utilisées pour la représentation graphique, mais elles ne sont pas traitées dans cette partie.

```
matde<-dist(matrice_documents, method = "euclidean")
```

```
hcTree = hclust(matde, "average")
```

# La fonction hclust permet de préciser le critère d'agrégation. On peut mettre complete , Ward, ou encore single à la place de average. Chacune d'elle produit un dendrogramme différent.

hcTree \$height : permet de donner la valeur des sauts d'inertie.

plot(hcTree) : Cette fonction permet d'afficher le dendrogramme. On peut ajouter d'autres paramètres.

Vous pouvez également utiliser la méthode agnes à la place de hclust ou diana (voir sections précédentes), comme suit :

```
> hcagnes <- agnes(matde, method = "average")
> hcagnes$height
[1] 6.146395 4.472136 4.990959 6.177169 6.645826 4.898979 6.077754 6.868793
[9] 7.066370 5.477226 5.523204 4.582576 4.943284 5.204455 4.582576 5.844617
[17] 5.930492 6.052728 6.159373 6.352959 7.802122 5.099020 5.611236 3.872983
[25] 6.115677 6.612639 6.662134 6.774242
> pltree(hcagnes, cex = 0.7, hang = -1, main = "Dendrogram of agnes")
> #pour utiliser la méthode plot, il faut transforemr le résultat de
agnes # à un hclust comme suit:
> treeAgnes<-as.hclust(hcagnes)
> plot(treeAgnes, cex=0.7,hang=-1)
>
```

---

<sup>14</sup> <https://cran.r-project.org/web/packages/factoextra/index.html>

```

> # méthode diana
> hcdiana<-diana(matde)
> hcdiana$height
[1] 6.782330 4.472136 5.291503 6.928203 7.211103 7.874008 4.898979 6.324555
[9] 9.273618 5.196152 6.082763 4.582576 5.196152 5.196152 5.656854 6.244998
[17] 6.324555 6.557439 6.708204 6.782330 8.544004 5.099020 6.324555 3.872983
[25] 6.557439 6.855655 6.928203 7.280110
> pltree(hcdiana, cex = 0.6, hang = -1, main = "Dendrogram of diana")

```

Notons pour chaque cluster formé, on appelle parangon l'individu dont les coordonnées sont les plus proches du centre de gravité de ce cluster. Il permet de caractériser le cluster auquel il appartient.

Essayons maintenant d'obtenir trois classes, ensuite quatre classes avec HCA. Pour ce faire, il suffit de découper l'arbre en utiliser les fonctions suivantes, une explication est donnée juste après :

```

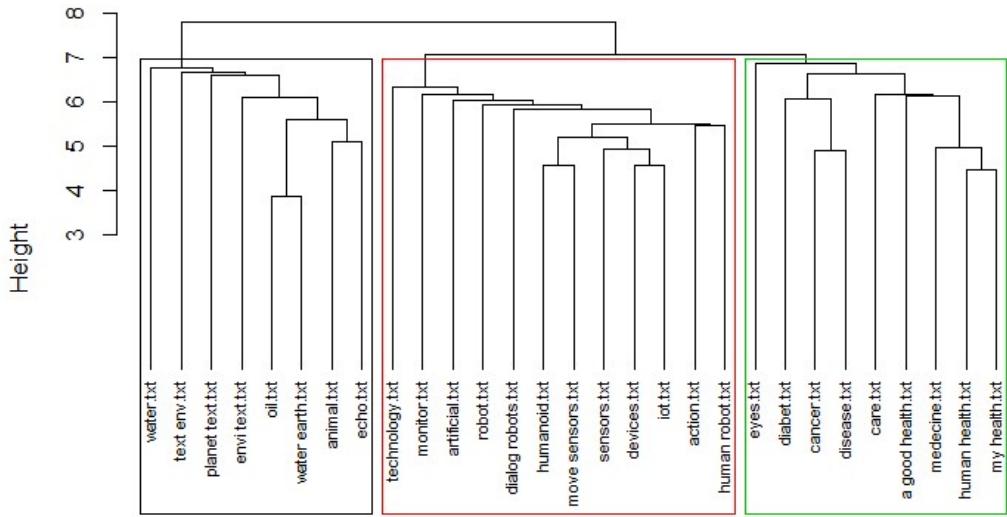
> plot(hcTree, cex=0.7,hang=-1)
> rect.hclust(hcTree, k = 3, border = 1:5)
> myclasses <- cutree(hcTree , k = 3)
> table(myclasses)
myclasses
 1  2  3
 9 12  8
> fviz_cluster(list(data = matde, cluster = myclasses))
> fviz_nbclust(matrice, FUN = hccut, method = "wss")
|
>

```

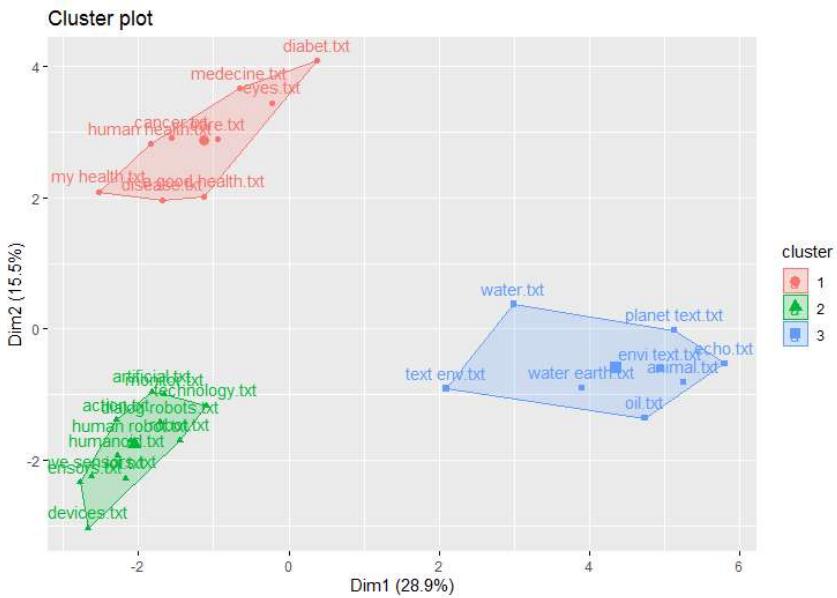
La première commande dessine l'arbre (dendrogramme) tel que *hang*=-1 pour mettre tous les noms des individus au même niveau, *cex* pour baisser la casse. La fonction *cutree*, renvoie un vecteur avec des appartennances à un groupe si *k* ou *h* sont scalaires, sinon une matrice avec des appartennances à un groupe est renvoyée où chaque colonne correspond aux éléments de *k* ou *h*, respectivement (qui sont également utilisés comme noms de colonne). Enfin, les deux dernières sont déjà exposées pour le choix de meilleur *k* cluster.

Les figures suivantes illustrent le résultat de ces commandes.

### Cluster Dendrogram



matde  
hclust (\*, "average")



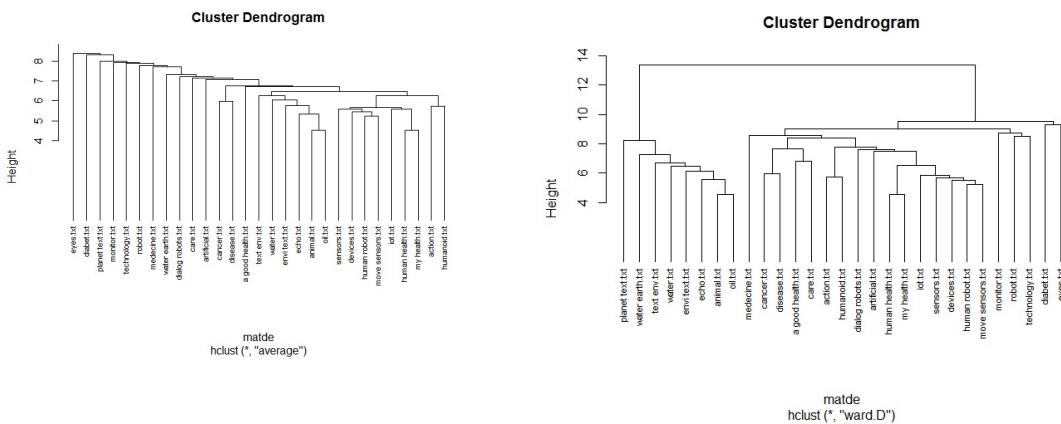
### Exercice

Créer quatre classes avec HCA, comparer le résultat avec celui de K-means.

### Exercice

Réaliser le travail en utilisant les différentes mesures de similarité étudiées jusqu'ici.

Retour sur l'algorithme HCA et le critère de ward. J'avais exposé que le critère de ward est utile pour supprimer l'effet de chaîne. Pour comprendre de quoi s'agit-il, vous allez dans un premier temps créer une matrice de distances basée sur la pondération TF.IDF. Ensuite, créer un dendrogramme HCA en utilisant le critère de saut *average* et la mesure de distance Euclide. Enfin visualiser le dendrogramme. Remarquer le résultat (figure à gauche). Pour corriger cet effet de chaîne, utilisant le critère de ward. Pour ce faire, il suffit de modifier le nom de la méthode *average* en *ward.D* comme suit : `hcTree = hclust(matde, method="ward.D")`



### **Mais, la question qui reste toujours posée, doit on utiliser la similarité Syntaxique ou sémantique ?**

Les mesures de similarités de textes ont été étudiées au début des années 70. Toutefois, il est sans doute évident que la similitude lexicale n'est pas du tout la même chose que la similitude sémantique. À titre d'exemple, basé sur le principe bacs of words, la phrase « Le patient est souffrant » n'a aucun lien avec la phrase : « Le malade est hospitalisé ». Bien que ces deux phrases soient sémantiquement similaires, il est presque sûr que des mesures lexicales existantes ne peuvent pas découvrir ce lien. Ainsi, le besoin d'extraire automatiquement des informations et d'effectuer une analyse sémantique est un domaine de recherche est devenu très actif lors de la conférence MUC (Message Understanding Conference) en 1987<sup>15</sup>. Il s'agit ici

---

<sup>15</sup> <https://www.aclweb.org/anthology/C96-1079.pdf>

d'extraire des relations logiques ou sémantiques entre des mots d'un texte. En outre, la reconnaissance des entités nommées (NER :Name Entity Recognition, en Anglais) entre autres peut déterminer le contexte sémantique et en extraire une information textuelle. A titre d'exemple, le terme *Dr* devrait désigner une personne (un humain) dont la profession est associée à la médecine. L'utilisation des règles, des approches statistiques, probabilistes et ontologiques peuvent aider à découvrir ces entités.

En vue de garantir une représentation sémantique et reproduire fidèlement le sens des textes, les méthodes linguistiques utilisent des techniques du traitement du langage naturel. Ces méthodes font appel également à des ressources externes telles que des dictionnaires, des ontologies, etc. l'objectif est d'obtenir des connaissances sur le domaine. Ces connaissances peuvent être le contexte dans lequel un texte est rédigé, le but recherché, les acteurs, la localisation géographique des auteurs et du thème étudié. En outre, il s'agit également s'imprégner des connaissances afin d'aboutir au même raisonnement humain. Ainsi, il est utile de détecter des relations (associations) entre concepts/termes, détecter des relations contextuelles internes aux textes afin de construire le sens des énoncés, le nombre de documents incluant des termes communs, etc. l'utilisation de POS tagging qui est une annotation grammaticale des mots basé sur le contexte d'apparition de ces mots permet de catégoriser des mots en fonction de leur rôle dans la phrase [10].

Les méthodes ne réduisent pas la dimension de l'ensemble de l'espace des fonctions mais suppriment simplement les composants «sans importance» des vecteurs de document individuels. Étant donné que la complexité du calcul de la similitude entre les documents est proportionnelle au nombre de composants non nuls dans les vecteurs de document, une telle troncature est efficace. En pratique, les vecteurs de documents eux-mêmes sont déjà assez clairsemés et seuls les centroïdes, qui peuvent être très denses, ont besoin d'être tronqués.

L'approche alternative est une réduction de la dimension globale. Son inconvénient est qu'il ne s'adapte pas aux caractéristiques uniques de chaque document. L'avantage est que cette méthode préserve mieux la capacité de comparer des documents différents car chaque document subit une transformation identique. Une des techniques

de réduction de dimension de plus en plus populaire est basée sur l'analyse sémantique latente (LSA, Latent Semantic Analysys). Ses applications sont variées telles que la recherche d'information, l'automatisation de processus de synthèse de textes, la catégorisation et la classification de textes, découvrir des relations entre les textes. En effet, l'étude de mesures des occurrences des mots est une des tâches de l'analyse sémantique latente. Contrairement à n-gram, LSA ne prend pas en compte l'ordre d'apparition des termes. Elle s'occupe uniquement à l'extraction des relations contextuelles de l'usage des mots dans un document. Ce qui permet d'établir une sorte de sémantique entre termes, associer des mots clés (nous pouvons parler de concept) à des documents.

Nous aborderons ces thèmes dans un second document qui fera suite à celui-ci. Toutefois, je vais aborder dans ce suivi les n-grammes.

L'objectif des n-gramme (n-gram en Anglais) est de trouver des contextes (sens) en associant des termes. En effet, en langue française comme l'Anglais la composition des mots peut avoir un sens différent s'ils sont explicités individuellement, à titre d'exemple non limitatif : watching machine (machine à laver), take off (si on considère que off est un mot vide, on perd le sens de décollage en Français), human being (être humain en Français) perd son sens également si on vient à lemmatiser being en is). Le principe de n-gram consiste à trouver des conséquences de termes consécutifs. Ainsi, un bi-gram associe deux termes, trois termes si on parle de tri-gram., etc.

## Manipulation de n-gram avec R

Puisque de nombreuses librairies coexistent, il se peut qu'un conflit puisse exister lors de l'utilisation d'une des fonctions (ce n'est pas toujours le cas, mais ça peut se produire). Pour remédier à ce désagrément, il est fort probable d'être obligé de détacher une librairie donnée pour pouvoir utiliser les fonctions d'une autre librairie. Par exemple, si vous voulez utiliser la fonction `tokens` de `quanteda`, il faut détacher la librairie `textstem`. Si vous n'êtes pas convaincu, comparer les deux figures ci-dessous. La première, montre le chargement des deux librairies `quanteda` et `textstem`. Comme le montre l'onglet de saisie de commandes, l'assistant qui propose automatiquement des fonctions disponibles pour une librairie ne fournit pas le choix de la fonction `tokens` de

la librairie `kropus`. La seconde figure quant à elle montre l'effet de détacher la librairie `textstem`. Si vous faites ce teste vous allez vous rendre compte que seule la fonction `tokens` associée à la librairie `korpus` soit proposée.

The image contains two side-by-side screenshots of the RStudio Source Editor. Both screenshots show the same R code in the editor pane:

```

1 library(quanteda)
2 library(textstem)
3 tokens

```

In the top screenshot, the cursor is over the word `tokens` in line 3. A tooltip appears in the bottom right corner of the editor pane, providing information about the function:

`tokens_lookup(x, dictionary, levels = 1:5, valuetype = c("glob", "regex", "fixed"), case_insensitive = TRUE, capkeys = !exclusive, exclusive = TRUE, nomatch = NULL, nested_scope = c("key", "dictionary"), verbose = quanteda_options("verbose"))`

The tooltip also includes the text: "Convert tokens into equivalence classes defined by values of a dictionary object." and "Press F1 for additional help".

In the bottom screenshot, the cursor is over the word `tokens` in line 3. A tooltip appears in the bottom right corner of the editor pane, providing information about the function:

`tokens`

The tooltip includes the text: "These methods return character vectors that return all types or token given text, where text can either be a character vector itself, a previous tokenized/tagged koRpus object, or an object of class koRpus.TTR." and "Press F1 for additional help".

Charger la librairie `qdap` (je suppose que vous avez déjà installé cette librairie). Si ce n'est pas le cas, installer-la avant de continuer. Ensuite, vérifier que la source des données est le répertoire contenant les fichiers que nous avons étudié jusqu'à présente. Enfin créer un data frame pour être utilisé par les fonctions de la librairie `quanteda`. La capture d'écran suivante résumé le travail qui doit être fait.

The screenshot shows the RStudio Source Editor with the following R code in the editor pane:

```

1 getwd()
2 matrice
3 library(qdap)
4 mydocs<-Corpus(DirSource(getwd()))
5 mydocsdataframe<-as.data.frame(mydocs)

```

The code is being typed into the editor, with the cursor currently on line 5. The status bar at the bottom left indicates "14:1" and the status bar at the bottom right indicates "R Script".

Visualiser le contenu de la variable `mysdocsdataframe` via l'onglet environment de RStudio. Cette variable est composée de deux valeurs intéressantes à savoir, `doc_id` et `text`. La première valeur représente l'intitulé de chacun des documents, quant à la seconde contient le contenu textuel. Ce qui nous permet d'utiliser les fonctions pour obtenir la liste des tokens et créer des matrices de n-grammes. Par conséquent, je vous laisse découvrir le résultat des commandes suivantes :

- `tokenlist<-tokens(mydocsdataframe$text)`
- `tokenlist.ngram<- tokens_ngrams(tokenlist, n=2, concatenator = " ")`
- Un concateneur permet d'indiquer le symbole tel que ( \_ au lieu de espace ) à utiliser pour associer deux token (termes).
- `tokenlist.ngram`

Vous n'aurez sûrement pas l'intégralité des résultats sur la console, ainsi, aller dans l'onglet environment et faire un clic sur le nom de la variable `tokenlist.ngram`. Comme le montre la figure ci-dessous, une icône sous forme de grille est affichée sur la même ligne de chaque texte, faire un clic sur cette grille pour afficher sur la console la totalité des tokens de chaque fichier.

| Name            | Type                   | Value                                                                                |
|-----------------|------------------------|--------------------------------------------------------------------------------------|
| tokenlist.ngram | list [29] (S3: tokens) | List of length 29                                                                    |
| text1           | character [41]         | 'people who' 'who exercise' 'exercise live' 'live longer' 'longer and' 'and keep ... |
| text2           | character [35]         | 'Sensors can' 'can measure' 'measure physical' 'physical properties' 'properties ... |
| text3           | character [71]         | 'These are' 'are renewable' 'renewable resources' 'resources because' 'because t ... |
| text4           | character [42]         | 'Now' ',' IoT' 'IoT applications' 'applications have' 'have the' 'the ability' ...   |
| text5           | character [55]         | 'As the' 'the future' 'future of' 'of healthcare' 'healthcare is' 'is closely' ...   |
| text6           | character [55]         | 'Health' ':' As' 'As officially' 'officially defined' 'defined by' 'by the' ...      |
| text7           | character [22]         | 'Robotic sensors' 'sensors are' 'are used' 'used to' 'to estimate' 'estimate a' ...  |
| text8           | character [56]         | 'Since the' 'the late' 'late 1970s' '1970s' ',', the' 'the federal' ...              |
| text9           | character [36]         | 'sensors help' 'help robots' 'robots to' 'to identify' 'identify the' 'the surro ... |
| text10          | character [35]         | 'Improving population' 'population health' 'health lies' 'lies in' 'in working' ...  |

Si vous avez perdu le contenu de la variable matrice des fichiers nettoyés, alors vous pouvez changer le répertoire de travail pour pointer vers les documents ayant subi des prétraitements. Première commande permet de changer de répertoire, la seconde

crée un nouveau corpus à partir des documents traités, et enfin la dernière commande crée le data frame. :

```
➤ setwd("C:\\\\Users\\\\sabri\\\\Documents\\\\docTraiter")
➤ mydocs<-Corpus(DirSource(getwd()))
➤ mydocsdataframe<-as.data.frame(mydocs)
```

## Exercice

Créer la matrice terme document avec 2-gram et ré-exécuter les algorithmes K-means et HCA. Que peut-on conclure ?

## Étude de similarité entre les documents.

Afin de profiter pleinement de la librairie quanteda, nous devons convertir la matrice en un objet reconnu par quanteda. Pour ce faire, la commande suivante charge de cette conversion : matrice\_dfm<-quanteda::as.dfm(matrice\_documents). A partir du résultat obtenu de la dfm, la fonction textstat\_simil calcule la similitude à partir des matrices de distances et de similitudes entre des documents ou des *features*. Pour ce faire, créer la matrice termes document à partir de vos fichiers nettoyés (ayant subi des prétraitement) : matrice\_documents <- DocumentTermMatrix(mydocs)

Vérifions maintenant la similarité entre les documents ou entre features avec la mesure cosine. Notons les seuils minimums. Modifier ces valeurs et vérifier manuellement en lisant les documents pour juger de l'efficacité de cette mesure.

```
> resultat_cosine<-textstat_simil(matrice_dfm, min_simil = 0.2, margin =
c("documents", "features"), method = "cosine")

> resultat_cosine<-textstat_simil(matrice_dfm, min_simil = 0.8, margin =
"features", method = "cosine")
```

Exécuter les commandes suivantes une à une. Le graphe des similitudes entre les textes de ce dendrogramme obtenu n'a rien avoir avec HCA.

```

plot(hclust(as.dist(resultat_cosine)))
as.matrix(resultat_cosine)
as.list(resultat_cosine)
as.list(resultat_cosine, diag = TRUE)
plot(hclust(as.dist(tstat4)))

```

Faire un Ctrl+L pour vider la console et gc() pour libérer de l'espace mémoire dans R. ensuite ré-exécuter les commandes suivante. Prenez le temps de lire les sorties sur la console.

```

(tstat4 <- textstat_dist(matrice_dfm, margin = "documents"))
as.matrix(tstat4)
as.list(tstat4)
as.dist(tstat4)
plot(hclust(as.dist(tstat4)))

```

## Bibliographie.

1. MARCUS, M. P., MARCINKIEWICZ, M. A., & SANTORINI, B. (1993). *Building a Large Annotated Corpus of English: The Penn Treebank*. In *Computational Linguistics*, 19(2), pages 313– 330.
2. Thomas Gaillat. *Annotation automatique d'un corpus d'apprenants d'anglais avec un jeu d'etiquettes modifiée du Penn Treebank*. TALN-RECITAL 2013, Jun 2013, France. pp.271-284, 2013.)
3. Jean Tournier . *Recherches en linguistique étrangère*. Vol. XIX. *Mélanges*. ISBN-10 2-251-60643-2 1996
4. Georges Kleiber . *Les différentes conceptions de la pragmatique ou pragmatique où es-tu ?*. In: *L'Information Grammaticale*, N. 1982
5. J. Courtes : *Introduction à la Sémiotique Narrative et Discursive*, 1976, coll. Hachette Université "Langue, Linguistique, Communication" [compte-rendu] sem-linkJacques Escande Pratiques Année 1978 20 pp. 109-111
6. LOVINS J. B. (1968) : Development of a Stemming Algorithm, Mechanical Translation and Computational Linguistics, 11 (1–2), pp. 22–31.
7. PAICE, C. (1996) : "Method for evaluation of stemming algorithms based on error counting", Journal of the American Society for Information Science, 47 (8), pp. 632– 349.
8. PORTER, M. (1980) : "An algorithm for suffix stripping", Program, 14 (3), pp.130–

137).

9. M. Paternostre et al. P. Francq, J. Lamoral, D. Wartel et M. Saerens. Juillet 2002. Carry, un algorithme de désuffixation pour le français
10. The Text Mining Handbook. Advanced Approaches in Analyzing Unstructured Data Ronen Feldman, Bar-Ilan University, Israel , James Sanger, ABS Ventures, Boston, Massachusetts 2006-2007. ISBN 9780511546914
11. G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975)
12. Automatic structuring of text files1 GERARD SALTON, CHRIS BUCKLEY AND JAMES ALLAN VOL. 5(1), 1–17 (MARCH 1992)
13. G. Salton, Automatic Text Processing—The Transformation, Analysis, and Retrieval of Information by Computer, Addison-Wesley Publishing Co., Reading, MA, 1989
14. Maron, M. E. (1961). “Automatic Indexing: An Experimental Inquiry.” *Journal of the Association for Computing Machinery* 8(3): 404–417.
15. Schutze, H., Hull,D. A., and Pedersen, J.O. (1995).AComparison of Classifiers and Document Representations for the Routing Problem. In Proceedings of SIGIR-95, 18th ACM InternationalConference on Research and Development in Information Retrieval. E. A. Fox,P. Ingwersen, and R. Fidel, eds. Seattle, ACM Press, New York: 229–237.
16. Wiener,E.D., Pedersen, J.O., andWeigend, A. S. (1995).ANeural Network Approach to Topic Spotting. In Proceedings of SDAIR-95, 4th Annual Symposium on Document Analysis and Information Retrieval. Las Vegas, ISRI, University of Nevada, Las Vegas: 317–332.
17. Wiener, E. D. (1995). A Neural Network Approach to Topic Spotting in Text. Boulder, CO, Department of Computer Science, University of Colorado at Boulder.
18. Forgy, E. W. (1965). Cluster analysis of multivariate data: efficiency vs interpretability of classifications. *Biometrics*, 21, 768–769.
19. Hartigan, J. A. and Wong, M. A. (1979). Algorithm AS 136: A K-means clustering algorithm.*Applied Statistics*, 28, 100–108. doi: 10.2307/2346830.
20. Lloyd, S. P. (1957, 1982). Least squares quantization in PCM. Technical Note, Bell Laboratories. Published in 1982 in *IEEE Transactions on Information Theory*, 28, 128–137.
21. MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, eds L. M. Le Cam & J. Neyman, 1, pp. 281–297. Berkeley, CA: University of California Press.