TP2 : Données textuelles

ENSTA

2021/2022

Adresse mail: mohammed.sedki@universite-paris-saclay.fr Page web: masedki.github.io

---

Des librairies

Au fil des questions, nous allons avoir besoin d'un ensemble de librairies R. Le code suivant permet de les installer et de les appeler automatiquement.

```
# les packages que nous allons utiliser
.packages <- c("tm", "SnowballC", "wordcloud", "caret", "rpart", "rpart.plot",
               "RandomForest", "gbm", "xgboost")

# verifier si un package est disponible sinon installer
.inst <- .packages %in% installed.packages()
if(length(.packages[!.inst]) > 0) {
  install.packages(.packages[!.inst], repos = "http://cran.rstudio.com")
}
# appel des librairies
lapply(.packages, require, character.only=TRUE)
```

I. Natural Language Processing (NLP)

Nous abordons ici quelques techniques de traitement automatique du langage naturel (NLP) pour mettre en œuvre des approches dites **linguistiques et terminologiques** pour le traitement de texte. Par exemple, la *tokenisation* des mots, le *stemming* (mise en évidence) des mots et la (pondération des mots) *term frequency-inverse document frequency (tf-idf) weighting*.

Il existe des librairies R pour le NLP. Ici, nous allons utiliser les packages tm (pour le traitement de texte) et SnowballC (pour le traitement des mots). Il existe des bibliothèques libres plus puissantes telles que Natural Langauge Toolkit (NLTK) et spaCy pour le NLP sous Python.

Nous avons téléchargé 431 notes et rapports cliniques accessibles au public à partir du *iDASH Clinical Notes and Reports Repository* pour construire des modèles d'apprentissage. L'ensemble de données a ensuite été annoté avec le thème de document approprié par deux cliniciens (Weng, 2017).

Lecture du jeu de données

Le jeu de données sous format Rdata se trouve ici.

```
# lecture des donnees (textes + labels)
load("idash.rda")
text_data <- idash$V1
text_label <- as.factor(idash$V2)
```

Avant d'ajuster un modèle d'apprentissage, le prétraitement et le nettoyage des données sont indispensables pour réduire les bruits et les informations inutiles. Nous effectuons généralement les étapes suivantes pour réaliser une *lexical normalization* :

1. convertir en minuscule
2. supprimer la ponctuation
3. supprimer les nombres
4. supprimer les blancs (espaces)
5. supprimer les *stopwords* (les mots communs comme *a, the, is, . . .*)
6. effectuer un stemming des mots (e.g. computers, computer, computing, compute -> comput)

La fonction `tm_map` du package tm permet d'appliquer les 6 étapes précédentes au corpus de texte constitué précédemment[1].

Après l'étape de **lexical normalization**, la prochaine étape consiste à générer la matrice appelée **document-term matrix** qui permettra l'ajustement d'un modèle d'apprentissage. Cette étape est relativement coûteuse en stockage mémoire (particulièrement sous R!). Pour contourner ce problème, la librairie `tm` nous fournit la fonction DocumentTermMatrix qui permet de générer une matrice documents-mots à l'aide des matrices dites **creuses**.

L'utilisation de la fonction `DocumentTermMatrix` avec les paramètres par défaut génère une matrice dite *bag-of-words* (sac de mots) avec la fréquence des mots (notée **tf**). Chaque ligne de la matrice correspond à un document et chaque colonne correspond à un mot unique (token) et la valeur correspondante sera le comptage d'occurrences du mot dans le document. R peut aussi nous générer une matrice dite pondération tf-idf, où les mots sont pondérés par leur importance dans l'ensemble du jeu de données.

7. Création de la matrice `dtm` de comptage d'occurrences des mots à l'aide de la fonction DocumentTermMatrix. Affichage des mots les plus fréquents (au moins 300 fois). On peut visualiser `dtm` à l'aide d'un nuage de mots

Pour commencer nous avons besoin de constituer le corpus de texte à l'aide du code suivant

```
corpus <- Corpus(VectorSource(text_data))
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeNumbers)
corpus <- tm_map(corpus, stripWhitespace)
corpus <- tm_map(corpus, removeWords, stopwords("english"))
corpus <- tm_map(corpus, stemDocument)

dtm <- DocumentTermMatrix(corpus)
```

[1] L'instruction vignette("tm") permet d'accéder à la vignette d'utilisation du package tm.

```
dtm <- DocumentTermMatrix(corpus,  control=list(wordLengths=c(3, 15),
                                                bounds = list(global = c(10,Inf))))
# document-term matrix is a sparse matrix
dtm
# # show the most frequent words
findFreqTerms(dtm, lowfreq=300)
```

```
# convertir une matrice creuse en data.frame
df <- suppressWarnings(data.frame(as.matrix(dtm)))
# tracer le nuage
wordcloud(colnames(df), colSums(df), max.words=50, min.freq=10,
          color=brewer.pal(6, "Dark2"), vfont=c("sans serif", "plain"))
```

8. Création de la matrice `dtm_tfidf` de pondération tf-idf à l'aide de la fonction DocumentTermMatrix.

```
dtm_tfidf <- DocumentTermMatrix(corpus,
                                control=list(bounds = list(global = c(35,Inf)),
                                             weighting=function(x)
                                               weightTfIdf(x, normalize=TRUE),
                                             stopwords=TRUE))
rownames(dtm_tfidf) <- 1:nrow(idash)
d <- data.frame(as.matrix(dtm_tfidf))
d$label <- label
```

À faire

1. À l'aide de la fonction createDataPartition, répartir le jeu de données en données apprentissage-test (70%-30%) comme suit :
   - `tfidf_train` contenant la tf_idf des documents sélectionnés pour le jeu de données d'apprentissage
   - `tfidf_train` contenant la tf_idf des documents sélectionnés pour le jeu de données de test
   - `label_train` contenant les étiquettes des documents du jeu de données d'apprentissage
   - `label_test` contenant les étiquettes des documents du jeu de données de test
2. Ajuster un arbre de décision pour expliquer l'étiquette du document (annotation des cliniciens) par la tf_idf (une sorte de variable explicative qui décrit le contenu du document). Afficher l'arbre de décision. Évaluer son erreur test.
3. Utiliser le package caret pour mettre en place un modèle de forêt aléatoire pour expliquer l'étiquette par la tf_idf du document.
4. Utiliser le package caret pour mettre en place un modèle de gradient boosting pour expliquer l'étiquette par la tf_idf du document. Utiliser différentes fonctions de perte. Penser à comparer les résultats obtenus à l'aide des packages gbm et xgboost.