Apprentissage statistique
ENSAE, 3ème année

Mohamed Hebiri

TP3: CRÉATION DE FEATURE AVEC WORD2VEC

Nous allons voir dans cette partie du TP comment on peut utiliser Word2vec sous `R`.

1 Préparation du répertoire et des fichiers

Le principe de ce TP est le même que ceux des TP précédents. Les étudiants sont invités à exécuter les commandes ligne par ligne en effectuant des copier-coller à partir du fichier pdf, mais en veillant à bien comprendre le but de chaque commande. Le temps estimé à passer sur ce TP est de 1,5h environ.

Il est fortement recommandé de sauvegarder toutes les commandes utilisées (en y ajoutant des commentaires si besoin) dans un fichier qui peut être nommé `TP3.R`. Le mieux est de commencer par changer le répertoire de travail en utilisant la commande `setwd` (set working directory).

On crée ensuite l'archive `texts.zip` dans lequel on va télécharger des textes qui se trouvent à l'adresse `http://mattmahoney.net/dc/text8.zip`. Ceci peut se faire soit avec la souris en allant sur la page et en téléchargeant le fichier dans le répertoire local, soit en ligne de commande en saisissant

```
file.create("texts.zip")
download.file("http://mattmahoney.net/dc/text8.zip", "texts.zip")
```

Attention, le fichier est de 30 Mo, le téléchargement peut prendre quelques minutes. Il faut ensuite décompresser l'archive:

```
unzip("texts.zip")
```

Pour vérifier que le fichier a été bien décompressé, vous pouvez lister tous les fichiers du répertoire de travail:

```
list.files()
```

Normalement, votre répertoire contient le fichier `text8` qui contient le texte qu'on va analyser. C'est un fichier de 100Mo environ.

2 Téléchargement et installation des paquets nécessaires

Nous allons utiliser une implémentation de Word2vec qui est disponible ici
`https://github.com/bmschmidt/wordVectors/`

Pour faire marcher cette implémentation, nous allons avoir besoin de `Rtools` correctement installé. Le téléchargement et l'installation de `Rtools` peut se faire à partir de ce site
`https://cran.r-project.org/bin/windows/Rtools/index.html`

Lancer ensuite les commandes suivantes[1]

```
# installer des paquets
  install.packages("devtools")
  install.packages("httr")
  install.packages("tm")
# charger des paquets
  library(devtools)
  library(httr)
  library(tm)
# configurer le proxy
  set_config(
    use_proxy(url="proxy.bloomberg.com", port=80)
  )
  set_config( config( ssl_verifypeer = 0L ) )
# Installer et charger wordVectors
  install_github("bmschmidt/wordVectors", INSTALL_opts = c('--no-lock'))
  library(wordVectors)
```

Sur mon PC, cela n'a pas fonctionné avec la version 3.3.2 de R. L'installation de la version 3.5.2 a résolu le problème.

3 Une commande prêt-à-utiliser

Rappelons que le but de `Word2vec` est d'associer un vecteur numérique à chaque mot d'un corpus de texte, de telle sorte qu'on puisse ensuite calculer des similarités entre les mots. Le but est de pouvoir afficher, par exemple, des listes de ce type:

```
                word similarity to "communism"
1          communism                 1.0000000
2          socialism                 0.8106134
3            marxism                 0.7555939
4          communist                 0.7512720
```

Le paquet qu'on a chargé, `wordVectors`, contient la commande

```
train_word2vec
```

qui prend en entrée le texte d'apprentissage et les paramètres d'entraînement, et qui fournit en sortie les vecteurs correspondant à chaque mot. Afin de simplifier l'utilisation de cette fonction, on peut utiliser une autre fonction

```
word2vec
```

dont le code se trouve ci-dessous (il faut le copier-coller dans R pour pouvoir l'utiliser). Elle va soit entraîner un word2vec (si fichier en entrée est un `.txt`) avec des paramètres par défaut, soit lire les vecteurs feature dans le fichier (si fichier en entrée est un `.bin`).

[1] La suite de ce TP utilise les codes de Kory Becker disponible à `https://gist.github.com/primaryobjects/8038d345aae48ae48988906b0525d175`

```
# Une fonction simple pour entraîner un word2vec à partir de file.txt ou
# pour charger un modèle existant du fichier file.bin.
  word2vec <- function(fileName) {
    if (grepl('.txt', fileName, fixed=T)) {
    # Convert test.txt to test.bin.
     binaryFileName <- gsub('.txt', '.bin', fileName, fixed=T)
    }
    else {
      binaryFileName <- paste0(fileName, '.bin')
    }

    # Train word2vec model.
      if (!file.exists(binaryFileName)) {
    # Lowercase and setup ngrams.
      prepFileName <- 'temp.prep'
      prep_word2vec(origin=fileName, destination=prepFileName,
        lowercase=T, bundle_ngrams=2)

    # Train word2vec model.
      model <- train_word2vec(prepFileName, binaryFileName,
        vectors=200, threads=4, window=12, iter=5, negative_samples=0)
    # Cleanup.
      unlink(prepFileName)
      } else {
      model <- read.vectors(binaryFileName)
    }
  model
  }
```

4 Un exemple simple

Commençons par nous échauffer sur un exemple simple.

```
# Télécharger un petit fichier de texte
  file.create("article.txt")
  site1 = "https://raw.githubusercontent.com/"
  site2 = "hebiri/TP3_ENSAE/master/text.txt"
  site = paste(site1,site2,sep="")
  download.file(site, "article.txt")
# Lire le fichier txt
  doc <- readChar('article.txt', file.info('article.txt')$size)
```

Nous allons ensuite supprimer les “stop-words”:

```
# Supprimer les ‘‘stop-words’’
  stopwords_regex <- paste(stopwords('en'), collapse = '\\b|\\b')
  stopwords_regex <- paste0('\\b', stopwords_regex, '\\b')
  doc <- stringr::str_replace_all(doc, stopwords_regex, '')
# Sauver dans un fichier le texte sans les ‘‘stop-words’’
  cat(doc, file="article2.txt",sep="\n",append=TRUE)
```

On peut remarquer que la commande `stopwords` est utilisée pour fournir la liste des “stop-words”. Cette commande fait partie du paquet `tm` que nous avons chargé au préalable. L'argument `'en'` de cette commande indique que nous nous intéressons au “stop-words” de l'anglais. On pourrait bien-sûr utiliser l'argument `'fr'` pour le français. Pour s'en convaincre et visualiser les mots considérés comme “stop-words” en français, il suffit de saisir la commande `stopwords('fr')`.

```
# Entraîner un modèle word2vec et l'explorer.
  start_time <- Sys.time()
  model <- word2vec('article2.txt')
  end_time <- Sys.time()
  print(end_time - start_time)
  model %>% closest_to("research")
# Cleanup.
  unlink('article2.txt')
```

5 Gros document

Pour limiter le temps d'attente, nous éviterons en classe d'analyser le document `texts` qu'on a téléchargé et décmpressé en début de séance. Je vous conseille d'ouvrir le fichier `texts` en utilisant un éditeur de texte quelconque (par exemple, bloc-notes) et supprimer une grosse partie du texte. Le fichier allégé[2], peut être sauvegardé sous le nom `text8small.txt`, par exemple.

5.1 Apprentissage du modèle

Nous commençons par entraîner le modèle `word2vec`, qui va associer un vecteur à chaque mot:

```
# chronométrer l'exécution (<4 min sur ma machine)
  start_time <- Sys.time()
    model <- word2vec('text8small.txt')
  end_time <- Sys.time()
  end_time - start_time
# exploration
  model %>% closest_to("communism")
```

[2] J'ai déjà effectué cette opération et déposé le fichier allégé sur pamplemousse. Vous pouvez le télécharger directement.

Chez vous, quand vous serez moins limité par le temps, effectuez l'apprentissage sur l'ensemble du corpus `texts`.

5.2 Affichage des projections

Nous pouvons maintenant projeter les vecteurs dans un plan pour visualiser le résultat. Le code ci-dessous propose de projeter les mots les plus similaires à "computer" et "internet" sur le plan engendré par les vecteurs associés à ces deux mots.

```
# Récuperer les vecteurs de  "computer" et "internet"
  computers <- model[[c("computer","internet"),average=F]]
# Récupérer les 3000 mots les plus fréquents et calculer
# la similarité avec "computer" et "internet"
  computer_and_internet <- model[1:3000,] %>% cosineSimilarity(computers)
  head(computer_and_internet)
# Filtrer en choisissant les 20 mots les plus similaires.
  computer_and_internet <- computer_and_internet[
  rank(-computer_and_internet[,1])<20 |
  rank(-computer_and_internet[,2])<20,
  ]
  plot(computer_and_internet,type='n')
  text(computer_and_internet,labels=rownames(computer_and_internet),cex = 0.7)
```

5.3 Clustering des mots

Nous pouvons utiliser la représentation vectorielle des mots pour effectuer un clustering. Cela peut se faire en utilisant la commande `kmeans` comme suit:

```
set.seed(10)
centers = 150
clustering = kmeans(model,centers=centers,iter.max = 40)
```

Comme le nombre de mots à regrouper est assez élevé (> 22000), on utilise un nombre de clusters relativement grand, égale à 150 ici.

Pour afficher le contenu du cluster 1, il suffit de saisir:

```
print(names(clustering$cluster[clustering$cluster==1]))
```

Vous obtenez un résultat similaire à celui-ci (mais pas nécessairement le même, car l'énumération des clusters se fait de manière aléatoire):

```
 [1] "arabic"          "letters"          "alphabet"         "letter"
 [5] "written_in"      "script"           "spelling"         "vowel"
 [9] "sounds"          "vowels"           "consonant"        "aramaic"
[13] "consonants"      "alphabets"        "the_arabic"       "syllable"
                       ...
[85] "ya"              "attracts"         "arabic_alphabet"  "tamil"
[89] "syllabary"       "cyrillic"
```

5.4 Réduction de dimension

Il existe une méthode de réduction de dimension globale intégrée à la bibliothèque pour visualiser les mots dans un seul plan globalement décent: réduction de dimension TSNE.

Le simple fait d'appeler "plot" affichera l'équivalent d'un nuage de mots avec des mots individuels groupés relativement proches les uns des autres en fonction de leur proximité dans l'espace de dimension supérieure.

La "perplexity" est le nombre optimal de voisins pour chaque mot. Par défaut c'est 50; des nombres plus petits peuvent amener les clusters à apparaître de manière plus dramatique au détriment de la cohérence globale.

```
library("tsne")
plot(model,perplexity=50)
```

A faire pour le compte-rendu du TP3

Utilisez votre éditeur de texte préféré (Word, LaTeX, Open Office) pour rédiger le compte-rendu. Convertissez le fichier final en **pdf** avant de l'envoyer à votre chargé de TD.

1. Trouvez un texte sur internet et appliquez à celui-ce ce que vous avez vu lors de ce TP. Vous pouvez également transformer le modèle obtenu en data-frame pour pouvoir l'utiliser simplement comme une entrée d'algorithmes d'apprentissage. Pour cela, il suffit de saisir

   ```
   features = data.frame(model)
   ```

2. Discutez de la pertinence des résultats obtenus.

La date limite pour l'envoi du compte-rendu est le 9 décembre 2019 inclus.