S10

2024-11-12

Read the basic packages

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   1.0.0
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.5.0
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(zoo)
```

```
##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(text2vec)
library(Rtsne)
```

Create a corpus with French UD data

```
# take a smaller list if needed
Languages <- c("French")

# open an empty table to store the output
data <- NULL %>% as.data.frame()

# extract data for the list of languages
for(z in c(1:length(Languages))){
  # take all the files in the folder of that language
  files <- list.files(paste("data_raw/UD/",Languages[z],"/",sep=""))
  # create and view an object with file names and full paths
  f <- file.path("data_raw/UD/",Languages[z], files)
  d <- lapply(f, FUN = function(files){read.delim(files,
                                                  header = FALSE,
                                                  comment.char = "#",
```

```r
                                                        stringsAsFactors = FALSE)})
  # combine all the files that were read
  merge.data <- plyr::rbind.fill(d)
  # add the language annotations
  merge.data <- merge.data %>%
    mutate(Language = Languages[z])
  # merge with the entire data
  data <- rbind(data, merge.data)
}
# remove not used vectors
rm(merge.data, d)

# arrange the columns of the table
data <- data %>%
  select(ID_word = V1, Tag = V6, POS = V4, Lemma = V3,
         Dependency = V7, Role = V8, Language)

# adding start and end of sentences
data <- data %>%
  # change IDs to numeric
  mutate(ID_word = as.numeric(ID_word)) %>%
  # add gap of IDs between consecutive pair of words
  mutate(diff = ID_word - lag(ID_word, default = first(ID_word))) %>%
  # change NAs to 0s if needed
  replace(is.na(.), 0) %>%
  # add labels
  mutate(ID_sentence = case_when(diff < 0 ~ "New_sentence",
                                 diff >= 0 ~ "In"))
```

```
## Warning in mask$eval_all_mutate(quo): NAs introduced by coercion
```

```r
# change new sentence markers to sentence number
data$ID_sentence[which(data$ID_sentence == "New_sentence")] <- 2:(length(data$ID_sentence[which(data$ID_
# manually add the start of the first sentence
data$ID_sentence[1] <- 1

# arrange the data
data <- data %>%
  # change the sentence ID to numeric
  mutate(ID_sentence = as.numeric(ID_sentence)) %>%
  # remove the diff column
  select(-diff)
```

```
## Warning in mask$eval_all_mutate(quo): NAs introduced by coercion
```

```r
# change NAs to the sentence ID
data$ID_sentence <- na.locf(data$ID_sentence)

# print the data as a table
data %>% write.csv("data_raw/UD.csv",
                   row.names = FALSE,
                   fileEncoding = "UTF-8")
```

```r
# extract the sentences
corpus <- data %>%
  # remove punctuation
  filter(!POS %in% c("_","PUNCT","")) %>%
  # take relevant columns
  select(Lemma, ID_sentence) %>%
  # group by sentence
  group_by(ID_sentence) %>%
  # put the words of the same sentence together in a column
  mutate(sentence = paste0(Lemma, collapse = " ")) %>%
  # take unique values in the sentence column
  pull(sentence) %>%
  unique()

# print the corpus in a file
corpus %>%
  writeLines("data_raw/corpus.txt")

# visualize the corpus
head(corpus)
```

```
## [1] "que être ce que un aide à le logement"
## [2] "pouvoir il avoir un aide à le logement"
## [3] "quel démarche devoir il effectuer pour avoir droit à un aide à le logement"
## [4] "pour quel type de logement pouvoir il bénéficier de un aide à le logement"
## [5] "à combien le/lui élever son aide à le logement"
## [6] "à_partir_de _ _ quand pouvoir il bénéficier de un aide à le logement"
```

Create the Glove embeddings. First, we create a vocabulary, i.e., a set of words for which we want to learn word vectors. These words should not be too uncommon. For example we cannot calculate a meaningful word vector for a word which we saw only once in the entire corpus. Here we will take only words which appear at least ten times. text2vec provides additional options to filter vocabulary (see ?prune_vocabulary).

```r
# Create iterator over tokens
tokens <- space_tokenizer(corpus)
# Create vocabulary. Terms will be unigrams (simple words).
it = itoken(tokens, progressbar = FALSE)
vocab <- create_vocabulary(it)
# Only keep vocabulary over a threshold
vocab <- prune_vocabulary(vocab, term_count_min = 10L)
# sanity check
tail(vocab)
```

```
## Number of docs: 31933
## 0 stopwords:  ...
## ngram_min = 1; ngram_max = 1
## Vocabulary:
##    term term_count doc_count
## 1:   un      14986     10731
## 2: être      15116     12954
## 3:    à      15312     10682
## 4:   de      45077     18520
```

```
## 5:   le      63911     21546
## 6:    _     176427      5909
```

Now we have terms in the vocabulary and are ready to construct term-co-occurence matrix (TCM), which we use to train the GloVe algorithm https://www.rdocumentation.org/packages/text2vec/versions/0.5.1/topics/GlobalVectors

```
# Use our filtered vocabulary
vectorizer <- vocab_vectorizer(vocab)
# create the tcm
tcm <- create_tcm(#tokenized corpus,
                  it,
                  # the vocabulary
                  vectorizer,
                  # the window
                  skip_grams_window = 5L,
                  # the direction of the window
                  skip_grams_window_context = "symmetric")
# use the tcm to train Glove
glove = GlobalVectors$new(#desired dimension for vectors
                          rank = 50,
                          #maximum number of co-occurrences used for weighting
                          x_max = 10)
# extract the vectors
wv_main = glove$fit_transform(tcm,
                              # number of iterations
                              n_iter = 10,
                              # set when does the model stop
                              convergence_tol = 0.01,
                              # number of cores to use
                              n_threads = 8)
```

```
## INFO  [08:33:26.946] epoch 1, loss 0.1830
## INFO  [08:33:27.574] epoch 2, loss 0.1068
## INFO  [08:33:28.125] epoch 3, loss 0.0885
## INFO  [08:33:28.693] epoch 4, loss 0.0774
## INFO  [08:33:29.268] epoch 5, loss 0.0693
## INFO  [08:33:29.889] epoch 6, loss 0.0632
## INFO  [08:33:30.558] epoch 7, loss 0.0585
## INFO  [08:33:31.097] epoch 8, loss 0.0546
## INFO  [08:33:31.645] epoch 9, loss 0.0514
## INFO  [08:33:32.196] epoch 10, loss 0.0487
```

We do a visual check to verify if the dimensions are correct. The numbers show the vocabulary size and the dimensions.

```
# combine main and context sets of word vectors due to mlapiDecomposition model
wv_context = glove$components
word_vectors = wv_main + t(wv_context)
dim(word_vectors)
```

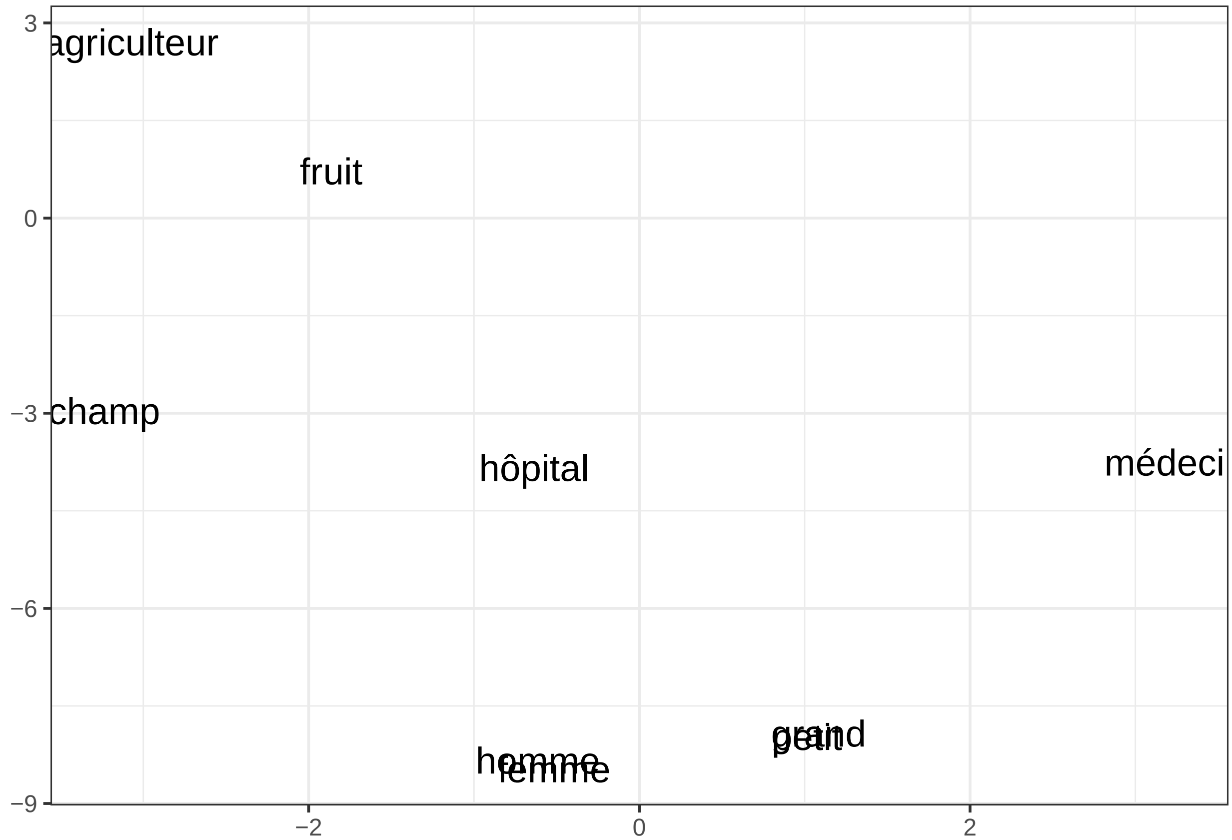
```
## [1] 4248   50
```

Then, we need to visualize the embeddings to see if they actually make sense or not. If we want to look at all the words in the corpus and how they relate to each other, there is a catchall method built into the library to visualize a single overall decent plane for viewing the library; TSNE dimensionality reduction (an intro to TSNE: https://www.analyticsvidhya.com/blog/2017/01/t-sne-implementation-r-python/). First, we transform the embeddings. This step might take a while (20-30 minutes) if the corpus we are using is quite big.

```
tsne <- Rtsne(word_vectors,
              # optimal number of neighbors for each word, reduce it if no clusters appear
              perplexity = 50,
              # change to TRUE if only plotting a small set of words
              pca = FALSE,
              check_duplicates = FALSE)
```

Now, we can visualize the words. We can choose to only show some words or all the words, in this example, I only show some of them to make the plot run faster.

```
tsne$Y %>%
  as.data.frame() %>%
  mutate(word = row.names(word_vectors)) %>%
  # select some words
  filter(word %in% c("homme","femme",
                     "grand","petit",
                     "fruit","champ",
                     "agriculteur","médecin",
                     "champ","hôpital")) %>%
  ggplot(aes(x = V1, y = V2, label = word)) +
  geom_text(size = 5) +
  theme_bw() +
  theme(axis.title=element_blank(),
        legend.title = element_blank())
```



Since the visualization shows that things are working properly, we can go on with measuring the semantic distance/similarity between words that are interesting for us. In the following examples, we measure the semantic similarity between words. The higher the similarity, the more similar two words are.

```
# set number of neighbors you want to see
number.of.neighbors = 30

# write a function
find_similar_words <- function(word, word_vectors, n = number.of.neighbors) {
  similarities <- word_vectors[word, , drop = FALSE] %>%
    # use the cosine similarity function
    sim2(word_vectors, y = ., method = "cosine")
  # rank the neighbors by similarity and take the top n
  similarities[,1] %>% sort(decreasing = TRUE) %>% head(n)
}

# examples
find_similar_words("femme",word_vectors)
```

```
##     femme     homme     jeune      chez   enfant  enceinte        et      tuer
## 1.0000000 0.7632976 0.7373868 0.6403120 0.5589599 0.5506139 0.4937842 0.4902555
##       ami   culture présenter       qui      sans     fille   intérêt politique
## 0.4871327 0.4691633 0.4668258 0.4567336 0.4441907 0.4376086 0.4347856 0.4342687
##  résultat    offrir    adulte     droit      avec       son  meilleur      site
## 0.4330994 0.4245948 0.4156383 0.4041705 0.4027659 0.3974167 0.3969276 0.3938554
##      pour        un      idée  fonction       âgé        ou
```

```
## 0.3918197 0.3910782 0.3906576 0.3901749 0.3901560 0.3859275
```

```r
find_similar_words("homme",word_vectors)
```

```
##      homme      femme  politique      jeune      droit         et         ou
##  1.0000000  0.7632976  0.7580445  0.7170471  0.6849398  0.6668992  0.6350347
##     enfant      autre    affaire   personne        qui  présenter       pour
##  0.6050574  0.5975786  0.5924389  0.5714397  0.5675098  0.5514739  0.5493998
##         un         le        sur         de       avec       site       tout
##  0.5489172  0.5428924  0.5279736  0.5274065  0.5222090  0.5047104  0.5017269
## constituer      trois       chez       tuer      celui       sans       dont
##  0.4916934  0.4916267  0.4873799  0.4841421  0.4752610  0.4707989  0.4693721
##    certain     marché
##  0.4598581  0.4582174
```