Textmining 3

- assessing the similarity of your own written coursework
- loading and working with word embeddings in R

Note: this tutorial assumes that you have installed the `stingdist` package and have downloaded the glove pretrained models.

Task 1: Text similarity and ngrams on your own coursework

Take a collection of your coursework in your study programme so far (or other coursework you stamped in). Read the files into R and create a corpus object from it.


```{r echo=TRUE}
#Here we load two word documents (word files for paper submissions)
#Best use the readtext package to read the files and then construct a corpus using the typical quanteda procedure.
library(readtext)
library(quanteda)
```

```{r}
doc1 = readtext('/Users/x/Dropbox/research/x/abusive_language/paper/lta_paper_jcss_submission.docx')
doc2 = readtext('/Users/x/Dropbox/research/x/gang_violence/paper/paper2/sentiment_drill_music_kleinberg_mcfarlane.docx')
```

```{r}
#Note that the text needed for the corpus is located in the `text` column
my_docs_corpus = corpus(c(doc1$text, doc2$text))
```

Use the data you have loaded to answer the following three questions:

1. Which unigram and which bigram did you use the most often in the whole corpus?

```{r}
unigrams = dfm(my_docs_corpus
      , tolower = T
      , remove = stopwords()
      , remove_punct = T
      , ngrams = 1)
bigrams = dfm(my_docs_corpus
      , tolower = T
      , remove = stopwords()
      , remove_punct = T
      , ngrams = 2)
```



```{r}
topfeatures(unigrams)
#Note that these ngrams are due to embedded meta data in the referencing.
```

```{r}
topfeatures(bigrams)
```

2. Of all documents, extract the first 50 words. Which documents have the greatest similarity on these first 50 words? Use the Levenshtein distance.

```{r}
words_50_doc1 = tokens(doc1$text, remove_punct = T)
words_50_doc1 = words_50_doc1$text1[1:50]
# for stringsim we need the words concatenated into one string:
string_words_50_doc1 = paste(words_50_doc1, collapse = ' ')
words_50_doc2 = tokens(doc2$text, remove_punct = T)
words_50_doc2 = words_50_doc2$text1[1:50]
string_words_50_doc2 = paste(words_50_doc2, collapse = ' ')
library(stringdist)
stringsim(string_words_50_doc1, string_words_50_doc2, method = 'lv')
```

3. Now that the last 20 words of each document and calculate the Euclidean-based similarity for all documents using q-grams with $q = 2$. Which two documents have the highest similarity? _*(Hint: you can best use a matrix to calculate similarity for all vectors at once)*_


```{r}
last_20_doc1 = tokens(doc1$text, remove_punct = T)
last_20_doc1 = last_20_doc1$text1[(ntoken(last_20_doc1)-
20):ntoken(last_20_doc1)]
string_last_20_doc1 = paste(last_20_doc1, collapse = ' ')
last_20_doc2 = tokens(doc2$text, remove_punct = T)
last_20_doc2 = last_20_doc2$text1[(ntoken(last_20_doc2)-
20):ntoken(last_20_doc2)]
string_last_20_doc2 = paste(last_20_doc2, collapse = ' ')
#note that we need the qgram output as a matrix to use the base R `dist` function
qgrams_output = qgrams(string_last_20_doc1, string_last_20_doc2, q = 2)
dist(qgrams_output, method = 'euclidean')
```

4. Does the cosine similarity confirm the finding from (3)?

```{r}
stringsim(string_last_20_doc1, string_last_20_doc2, q=2, method = 'cosine')
#We have only two documents here. For more than two you can check whether the order of proximity for the Euclidean measure is the same as the order of cosine similarity (i.e. are those who are closest to one another also the most similar?)
```

Task 2: Word embeddings with Glove in R

Word embeddings are among the most interesting developments in NLP in the past decade. Using them in R is relatively easy once you have done the setup right. The tricky part is the computationally heavy load of the pre-trained models (we do not cover custom training or re-training of our own models in this module).

We ran a tutorial on word embeddings at a computational social science workshop. Use this tutorial to start with word embeddings in R.

- Tutorial: https://raw.githack.com/maximilianmozes/word_embeddings_workshop_resources/master/code/R/glove_in_R.html - you can do that tutorial in the dedicated file.

Once you have done this, use the `fit_glove(...)` function to load different models with different dimensions.

Note: the bigger the model and the higher the dimensions, the bigger the file you will load into your computer's memory. Too big files will slow done your machine. If you have an 8 GB memory, you should be fine in using the 6B models (with all dimensions) and the 2B Twitter model. Do not use the 42B or higher models unless you have got at least 16 GB RAM.

Do the vector similarities for the following words differ if you load different models?

- crime
- machine
- girl
- lecture

```{r}
```

```r
source('/Users/x/GitHub/r_helper_functions/init_glove.R')
#for the 6B/100d model:
init_glove(dir = '/Users/x/Documents/glove', which_model = '6B', dim=100)
crime_100d = as.vector(glove.pt[row.names(glove.pt) == 'crime', ])
machine_100d = as.vector(glove.pt[row.names(glove.pt) == 'machine', ])
girl_100d = as.vector(glove.pt[row.names(glove.pt) == 'girl', ])
lecture_100d = as.vector(glove.pt[row.names(glove.pt) == 'lecture', ])
#use our cosine similarity function
cossim = function(A, B){
 numerator = sum(A*B)
 denominator = sqrt(sum(A*A))*sqrt(sum(B*B))
 cosine_sim = numerator/denominator
 return(cosine_sim)
}
#you can then compare the vectors pair-wise:
cossim(crime_100d, machine_100d)
cossim(crime_100d, girl_100d)
cossim(crime_100d, lecture_100d)
```



```{r}
#now we redo these steps with a different pretrained model
#for the 6B/100d model:
init_glove(dir = '/Users/x/Documents/glove', which_model = '6B', dim=50)
crime_100d = as.vector(glove.pt[row.names(glove.pt) == 'crime', ])
machine_100d = as.vector(glove.pt[row.names(glove.pt) == 'machine', ])
girl_100d = as.vector(glove.pt[row.names(glove.pt) == 'girl', ])
lecture_100d = as.vector(glove.pt[row.names(glove.pt) == 'lecture', ])
#use our cosine similarity function
cossim = function(A, B){
 numerator = sum(A*B)
 denominator = sqrt(sum(A*A))*sqrt(sum(B*B))
 cosine_sim = numerator/denominator
 return(cosine_sim)
}
#you can then compare the vectors pair-wise:
cossim(crime_100d, machine_100d)
cossim(crime_100d, girl_100d)
cossim(crime_100d, lecture_100d)
```


Do these findings surprise you?

Task 3: Using word embeddings

Use the word embeddings you loaded in Task 3 and answer these questions with the appropriate arithmetic operations:

- Which embedding is closer to "LONDON": "WESTMINSTER" or "ENGLAND"?
- What is the closest neighbour to the vector retrieved by calculating: "DOCTOR" - "MAN" + "WOMAN"? What did you expect?
- In the Twitter pre-trained model, which concept is closest in the vector space to "SUCCESS"?

```{r}
#1.
london_emb = as.vector(glove.pt[row.names(glove.pt) == 'london', ])
westminster_emb = as.vector(glove.pt[row.names(glove.pt) == 'westminster', ])
england_emb = as.vector(glove.pt[row.names(glove.pt) == 'england', ])
cossim(london_emb, westminster_emb)
cossim(london_emb, england_emb)
#Answer: "England" is closer to "London" than "Westminster"
```



```{r}
#2.
init_glove(dir = '/Users/x/Documents/glove', which_model = '6B', dim=100)
woman_emb = as.vector(glove.pt[row.names(glove.pt) == 'woman', ])
doctor_emb = as.vector(glove.pt[row.names(glove.pt) == 'doctor', ])
man_emb = as.vector(glove.pt[row.names(glove.pt) == 'man', ])
mystery_vector = doctor_emb - man_emb + woman_emb
#to calculate vector proximity withnthe glove.pt dfm, we need to add the mystery vector to the dfm:
mystery = data.frame(matrix(mystery_vector, nrow = 1, byrow = F))
names(mystery) = paste('vec_', 1:100, sep="")
q = as.dfm(mystery)
glove.pt2 = rbind(glove.pt, q)
row.names(glove.pt2)[400001] = c('mystery_1')
cos_sim_vals = textstat_simil(glove.pt2
             , selection = c("mystery_1")
             , margin = "documents"
             , method = "cosine")
```

```
head(sort(cos_sim_vals[,1], decreasing = TRUE), 10)
```

```{r}
cos_sim_vals = textstat_simil(glove.pt
             , selection = c("success")
             , margin = "documents"
             , method = "cosine")
head(sort(cos_sim_vals[,1], decreasing = TRUE), 50)
```

Task 4: (advanced) train word embeddings on your own mini-corpus

This task is related to and an extension of the homework Part 1 below.

Remember how we used pre-trained models so far? You can train your own models which might be a good option if you are interested in a more localised language phenomenon. A major downside is of course the size of the corpora used for training. Common sizes used in the provided pre-trained models are 840 billion documents, for example.

For now, collate a mini-corpus of texts that you have written. Create a `corpus` object.

```{r}
#Your code here
```

Now you can use that corpus to construct a co-occurrence matrix and embed the resulting vectors. To do this, follow the steps in the [quanteda text2vec](https://quanteda.io/articles/pkgdown/replication/text2vec.html) tutorial.