Word Embeddings

Thomas Asikis

11/27/2019

Introduction

This documens summarizes and explains some code snippets on how to generate word embeddings (or representations) using two popular algorithms **word2vec** and **GloVe**.

Related Data

This time we use a parliamentary corpus from:

https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/6MZN76. The corpus focuses on parliamentry debates of the Irish parliament. It can be acquired from dataverse, but we download it locally and put it in the same folder as the scripts.

Loading the data

The data are loaded as follows. Since generating word embeddings may take a lot of time, we only focus on available texts coming from the last goverment period, starting from 2011-03-09. Another reason to subsample the data based on period are the considerations on changing context of words throughout time.

```
library(data.table)

# again we set the working directory for R studio as the folder this script is in
# source.folder <- setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

# we collect the corpus and set the date column to the appropriate type
corpus <- data.table::fread("Dail_debates_1919-2013.tab", header = T)
corpus$date <- as.Date(corpus$date)

# we filter the corpus to get only a portion of the data.
last_goverment_begin_data <- as.Date('2011-03-09')
corpus <- corpus[corpus$date >= last_goverment_begin_data, ]
```

Preprocessing

As for preprocessing, we choose to remove stopwords, normalize characters to remove irish accents and convert the text to lowercase, before tokenization:

```
library(stringi)
library(stopwords)
library(tokenizers)

# normalization for accents
corpus$norm_speech <-  stri_trans_general(str = corpus$speech,
                                          id = "Latin-ASCII")
```

```
# now we tokenize the corpus
tokenized_corpus <- tokenize_words(corpus$norm_speech,
                                   stopwords = stopwords::stopwords("en"),
                                   lowercase = TRUE,
                                   simplify = TRUE)
```

Term Co-occurence Matrix

In the next step we use the `text2vec` to convert the tokens to a format compatible for processing with the GloVe algorithm. More specifically we want to get the term co-occurence matrix, which compare terms against each other and measures how ofter they appear together within a window in text. Here we choose to keep terms that appear more than 5 times (\texttt{term_count_min = 5L}) in the text and we also use a window of size 5 (\texttt{skip_grams_window = 5L}).

```
library(text2vec)

# text2vec token iterator
it = text2vec::itoken(tokenized_corpus, progressbar = FALSE)

# creation of vocabulary
vocab <- text2vec::create_vocabulary(it)

# removal of infrequent terms
vocab <-  text2vec::prune_vocabulary(vocab, term_count_min = 5L)
vectorizer <- text2vec::vocab_vectorizer(vocab)
it = text2vec::itoken(tokenized_corpus, progressbar = FALSE)

# term co-occurrence matrix creation
tcm <- text2vec::create_tcm(it, vectorizer, skip_grams_window = 5L)
```

Word2Vec Preprocessing

As we use a different library for the word2vec vectors, we need to diversify our preprocessing schedule as follows:

```
library(wordVectors)
# we unlist the corpus recursively to crate a huge chunk of text
w2v_corpus <- unlist(tokenized_corpus)
# we now put all data in a big list
w2v_corpus <- list(w2v_corpus)
# and persist all the tokens in a file
data.table::fwrite(w2v_corpus, file='corpus_text.txt',  sep=' ', quote=FALSE, eol=' ')

# prepare the word2vec corpus and write it to a file.
# here we consider single words as terms.
# ngrams = 2 would also consider as terms all the possible pairs of words as well
prep_word2vec(origin="corpus_text.txt",
              destination="w2v_corpus_text.txt",
              lowercase=T,
              bundle_ngrams=1)
```

```
## Beginning tokenization to text file at w2v_corpus_text.txt
```

```
## Prepping corpus_text.txt
```

Word Embeddings

Now that the data preprocessing is mostly finished we proceed to create the embeddings and save them in the local disk. Hyperparameter optimization could be done here as well for each algorithm, but for the sake of a quick example we just pick a random set of parameters.

GloVe embeddings

For GloVe algorithm, the following hyperparameters are taken into account: - \texttt{word_vector_size}, which is the number of dimensions for the word embedding vectors. Usually a higher number of dimensions indicates better information transfer to the embedding, but in practise this is not always the case. - \texttt{x_max} GloVe uses a weighting matrix based on term co-occurences to determine the word embedding values. This parameters determines the maximum number of co-occurences that will affect those weights. - \texttt{learning_rate}: In a broad sense parameter controls how much the value of the loss function affects the change of the parameter values. High loss values and learning rates make the parameters change more drastically. Still, this is not always good as good parameters values might be skipped if the change is too drastic. - `alpha`: a parameter that connects to the weighting function and \texttt{x_max}. Higher values of \texttt{x_max} and lower values of `alpha` push the term weights with a low number of co-occurences to 0, therefore reducing their effect on the embedding calculations. The GloVe model produces 2 sets of vectors, the main output of the model and the context vectors that are calculated withing the model. Either can be used, still it is common practice to sum or average elementwise between the vectors.

```
library(text2vec)
# hyperparameter setting
glove_model = GlobalVectors$new(word_vectors_size = 50,
                            vocabulary = vocab,
                            x_max = 10,
                            learning_rate = 0.1,
                            alpha = 0.75,
                            lambda = 0.0)

# the main vectors per term are created via fitting the model
main_vectors <- glove_model$fit_transform(tcm, n_iter = 10, convergence_tol = 0.01)

# in the model the context vectors are calculated
context_vectors <- glove_model$components

# it is a common practical trick to sum those.
# Still, the individual vectors may work better.
glove_vectors <- main_vectors + t(context_vectors)

# persist vectors to file.
write.table(glove_vectors, file = "glove_vectors_matrix", append = FALSE)
```

Word2Vec Embeddings

Below we train the word2vec model. To do so we use the text file generated in the preprocessing step. There are many parameters to consider in general, but for this package, the most important ones are: - `cbow`: whether to use the continuous bag of words or skipgram implementation for training. In general

skipgram is observed to work better. - `window`: The window of words to consider during training of the model. - \texttt{negative_samples}: the number of negative samples or 'wrong training samples' to use during algorithm training. Usually increasing this number may help in smaller corpora.

```
# train the model and persist it
library(wordVectors)
word2vec = wordVectors::train_word2vec("w2v_corpus_text.txt",
                        "w2v_corpus_text.bin",
                        vectors=50,
                        cbow=FALSE,
                        threads=4,
                        window=5,
                        min_count=5,
                        iter=5,
                        force=TRUE,
                        negative_samples=10
                        )
```

Since training may take a lot of time, once we persisted the model, we just load the file as follows:

```
library(wordVectors)
word2vec = wordVectors::read.vectors("w2v_corpus_text.bin")
```

```
## Filename ends with .bin, so reading in binary format
```

```
## Reading a word2vec binary file of 34934 rows and 50 columns
```

Using the models

Both word2vec and GloVe can be used for word reprensentations. To do so, one needs to request the model for the token and also the embedding dimensions of interest. For GloVe the word embedding of the token "funding" looks like:

```
glove_vectors['funding', ]
```

```
##  [1]  0.55933022  0.22056832  0.58588129 -0.19603911  0.38905963  0.87093183
##  [7]  0.15390350  1.54078341  0.36840975 -0.29715451 -0.13234275  0.41708300
## [13]  1.41313744  1.49526316  0.84950325 -0.64959943  0.69633028 -0.07715211
## [19]  0.54759313 -0.82005522 -0.35802501 -0.72727275  0.53503408  1.28829062
## [25] -1.26028931  0.08649709 -0.65282416 -0.53428963  0.53625494  0.15025171
## [31]  0.67055723  1.03835633 -0.60060892 -0.56122191 -0.05711289  0.47791737
## [37] -0.34041229 -0.80323535  0.86999109 -0.99467641  0.05881092  1.02513656
## [43]  0.97557762 -0.85586092  0.51037163  0.22082699  0.88337421 -0.89770865
## [49] -0.59153569 -0.14874898
```

For word2vec the word embedding of the token "funding" looks like:

```
##  [1] -0.124917150  0.231705874 -0.513827026  0.300119728 -0.094708577
##  [6]  0.186219454  0.229554534  0.139177486 -0.003654234 -0.157469660
## [11]  0.240254939 -0.282590657  0.306540191  0.319577366  0.244253248
## [16]  0.431352228  0.070256919 -0.213574111  0.172907412 -0.077534609
## [21] -0.136361897 -0.137386277  0.250961423 -0.030548781  0.023299256
## [26]  0.050375436 -0.535753012  0.002378043 -0.002653219 -0.074094698
## [31] -0.252944767 -0.441292077  0.294001549 -0.053392783 -0.399243772
## [36]  0.107505485 -0.113705702  0.833677292  0.650928080 -0.769376516
## [41]  0.210874483  0.002052813 -0.563713789 -0.436405867  0.062631004
## [46]  0.698130429  0.016358422 -0.103686541  0.477846354  0.351105005
```

One can also perform analogy tasks to check the contextual linking between words according to the model embeddings. E.g. What is the most probable word to be combined wth the token "capital", so that the contextual meaning is analogous to the combinations of tokens "funding" and "left" according to the word2vec model?

```
library(magrittr)
word2vec %>%
               wordVectors::closest_to(~"funding" - "right" + "left", n=2)
```

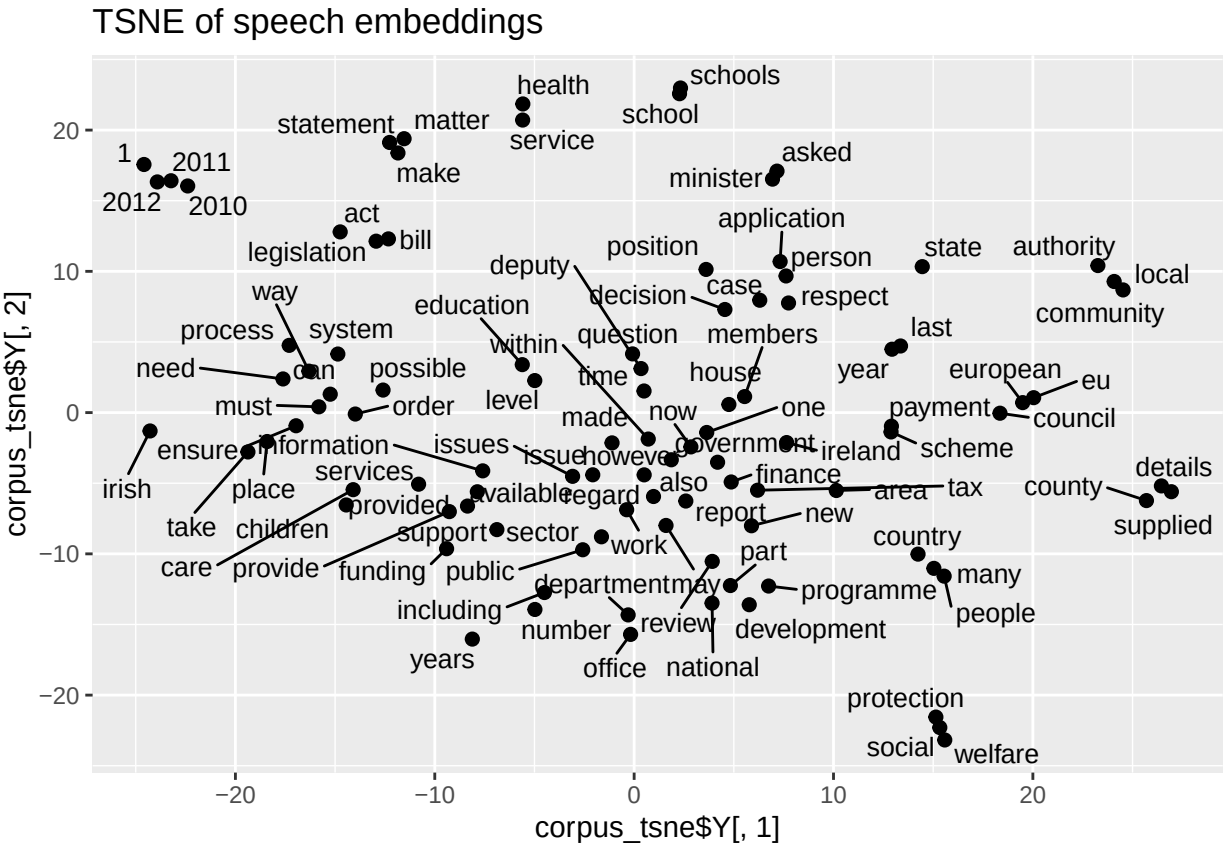
```
##        word similarity to "funding" - "right" + "left"
## 1 earmarked                                   0.7637959
## 2   funding                                   0.7632761
```

Plotting word embeddings

A common practise to study and undestand word embeddings is to plot them on a scatter plot and try understand contextual similarities. Usually this is done via some dimensionality reduction process (e.g. PCA) or by selecting 2 dimensions from the word embeddings in an exploratory manner. Below you can find a scatter plot of the most frequent tokens in the dataset, after doing a TSNE on the word embeddings.

```
library(Rtsne)
library(ggplot2)
library(ggrepel)
terms_interest <- vocab[order(-vocab$term_count), ][1:100, ]
word2vec_interest <- word2vec[terms_interest$term, ]
corpus_tsne <- Rtsne(word2vec_interest, dims = 2,
                     perplexity=4, verbose=FALSE, max_iter = 500, check_duplicates = FALSE)

ggplot(terms_interest, aes(x=corpus_tsne$Y[,1], y=corpus_tsne$Y[,2]), label=term) +
  geom_point(size=2) +
  geom_text_repel(aes(label = terms_interest$term), size = 3.5) +
  ggtitle("TSNE of speech embeddings")
```



What do you think of the outcome? What if one colored the tokens according to the party that most frequently uses it?