Assignment 2: Distributional Semantic Models

Eva Maria Vecchi
Lexicon, Syntax, Semantics IIb: Modeling Meaning

Due Date: June 5, 2020

Meaning and Distribution

1. In your own words, state some advantages DSMs have as an approach to model word meaning (discuss at least 2).
2. Based on what we have discussed until now, in your own words, describe any disadvantages or shortcomings with this approach (e.g. where might it fail, what elements of word meaning might not be captured, etc.).
3. What considerations/differences must you take into account when implementing this approach in English vs. Hungarian vs. Tagalog?

Building your own DSM

We will be using the `wordspace`[1] package in `R` for this assignment. It is an interactive laboratory for empirical research on DSMs. It consists of a small set of carefully designed functions, most of which (i) encapsulate non-trivial `R` operations in a user-friendly manner or (ii) provide efficient and memory-lean `C` implementations of key operations. To install the package:

```
> install.packages("wordspace")
```

4. Follow the `wordspace` tutorial. Build the DSM as described in the tutorial, and become familiar with the package and how to parameterize, visualize, and evaluate a DSM.
5. Follow the exercise[2] below and experiment with the parameters.
   a. Can you clearly identify "good" and "bad" parameter settings? Explain, providing examples.
   b. Determine a specific parameter setting (specify which you used and why) and build DSMs using different co-occurence contexts, for example both `WP500_Win5_Lemma` and `WP500_Win30_Lemma`. Describe how the quality of the neighborhoods differ (e.g. similarity vs. relatedness). Provide some examples.

[1]Evert, Stefan (2014). Distributional semantics in R with the wordspace package. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: System Demonstrations*, pages 110114, Dublin, Ireland.

[2]Evert et al (2018). Distributional Semantics – A practical introduction. *Introductory course at ESSLLI 2018.* Sofia, August 6-10, 2018. Part 2 exercise.

```
##
## Exercise for part 2:
## Use the wordspace package to construct DSMs with different parameters
##

library(wordspace)

## Several pre-compiled DSMs based on the English Wikipedia (WP500 corpus)
## using different co-occurrence contexts are available for download from
##
##     http://wordspace.collocations.de/doku.php/course:material#pre-compiled_dsms
##
## The following co-occurrence contexts are available:
##   TermDoc    ...  term-document matrix
##   Win30      ...  30-word span (L30/R30)
##   Win5       ...  5-word span (L5/R5)
##   Win2       ...  2-word span (L2/R2)
##   DepFilter  ...  dependency-filtered
##   DepStruct  ...  dependency-structured
##   Ctype_L1R1  ... L1+R1 context types (pattern of left & right word)
##   Ctype_L2R2  ... L2+R2 context types (left & right 2 words, very sparse)
##   Ctype_L2R2pos ... L2+R2 part-of-speech context types
##
## All models are available as raw co-occurrence counts with marginal frequencies,
## as well as in a pre-compiled version (log simple-ll, SVD to 500 dimensions, P = 0).

## In this exercise, we will use the raw co-occurrence matrix so that we can test
## different settings for all the DSM parameters.  Download one of the models, e.g.
##
##     WP500_Win5_Lemma.rda
##
## and save the file in your project directory (usually the same as this R script).
## If your computer has limited RAM, it may be better to pick one of the more specific
## co-occurrence contexts resulting in a smaller model (file size < 60 MB).

## Load the co-occurrence matrix and marginal frequency data
load("WP500_Win5_Lemma.rda", verbose=TRUE) # verbose option prints name of the DSM object

## It is convenient to assign the model to a shorter variable name
WP <- WP500_Win5_Lemma # R only makes a copy if you modify one of the two variables

## Take a first look at the data
WP # shows number of rows and columns as well as fill rate (-> sparseness)
head(WP, 15) # top left corner of co-occurrence matrix
head(WP$rows, 10) # row marginals (NB: f == 10 * frequency because of span-size adjustment)
head(WP$cols, 10) # column marginals (here f is the corpus frequency of the term)
## Terms in this model are POS-disambiguated lemmas with part-of-speech codes
## _N (noun), _V (verb), _J (adjective), _R (adverb)

## Now apply different scores, transformation normalizations, etc. to the model and
## take a look at nearest neighbours for selected words.  It would be best to always
## look at the same targets as you change parameters, so let's collect them in a vector
## (you might want to select a different set of targets, of course):
words <- c("dog_N", "book_N", "walk_V", "smile_V")

## E.g. PPMI scores with L3 normalization (for Minkowski p=3 distance):
WP <- dsm.score(WP, score="MI", normalize=TRUE, method="minkowski", p=3)

## nearest.neighbours() accepts multiple target words
nearest.neighbours(WP, words, n=10, method="minkowski", p=3)
## NB: method= and p= arguments are documented in ?dist.matrix

## Nearest-neighbour (NN) search among the rows of a sparse matrix is fairly slow for
## technical reasons.  If you run out of patience, here are two things you can do:

## 1) Compute NN for a large number of words you're interested in at the same time:
NN.list <- nearest.neighbours(WP, words, n=20, method="minkowski", p=3)
NN.list$dog_N         # now show individual neighbour sets from the list
NN.list[["smile_V"]] # safer in case a word contains non-standard characters

## 2) If your vesion of R supports multi-threading, try to activate it
wordspace.openmp()
wordspace.openmp(threads=4) # rule of thumb: number of physical CPU cores

nearest.neighbours(WP, words, n=10, method="minkowski", p=3)

## If you also want to experiment with dimensionality reduction, you may need to
## simplify your model by filtering rows and columns.  This will also speed up NN
```

```r
## search in the unreduced matrix, of course.

## The subset() function allows you to select rows and columns with the arguments
## subset= (rows) and select= (columns).  The filtering expressions are based on
## the variables term, f, nnzero from the row/column marginal tables.
## See the method documentation ?subset.dsm for further information and examples.

## For example, let us only look at verbs as targets and medium-frequency features
## with 400 <= f <= 10000 (based on the histogram below)
hist(log10(WP$cols$f), breaks=40) # 2 = 100, 3 = 1000, 4 = 10000, ...
abline(v=log10(c(400, 10000)), col="red")

WP2 <- subset(WP, subset=(grepl("_V$", term)), select=(f >= 400 & f <= 10000), update.nnzero=TRUE)
WP2 # considerably smaller now

WP2 <- dsm.score(WP2, score="MI", normalize=TRUE, method="minkowski", p=3) # need to renormalize rows (
    why?)

## Use skip.missing=TRUE to ignore target words not found in the model
nearest.neighbours(WP2, words, n=10, method="minkowski", p=3, skip.missing=TRUE)

## SVD dimensionality reduction usually applied after Euclidean normalization (=> part 5)
WP2 <- dsm.score(WP2, score="MI", normalize=TRUE) # default normalization is Euclidean

## SVD-based dimensionality reduction to n=100 latent dimension with whitening (P = 0)
WP100 <- dsm.projection(WP2, n=100, method="svd", power=0)
## NB: WP100 is a plain matrix (rows labelled with target terms), not a DSM object
## NB: 100 latent dimensions are often not enough for good semantics => try n=300 or n=500

nearest.neighbours(WP100, words, skip.missing=TRUE) # NN are much faster in reduced matrix

## If you want to experiment with power scaling, it's much faster to adjust post-hoc:
sigma <- attr(WP100, "sigma")
WP100.P1 <- scaleMargins(WP100, cols=(sigma ^ 1)) # because we started with P = 0

## You can use matrix indexing to skip latent dimensions
WP100.P1.skip <- WP100.P1[, 21:100] # skips first 20 latent dims

## Don't forget to re-normalize the vectors appropriately (unless you're using cosine/angle)
WP100.P1.skip <- normalize.rows(WP100.P1.skip, method="manhattan")
nearest.neighbours(WP100.P1.skip, words, n=10, method="manhattan", skip.missing=TRUE)
```