# Distributional Semantics in R with the 'wordspace' Package

*Stefan Evert*

*1 April 2016*

Distributional semantic models (DSMs) represent the meaning of a target term (which can be a word form, lemma, morpheme, word pair, etc.) in the form of a feature vector that records either co-occurrence frequencies of the target term with a set of feature terms (*term-term model*) or its distribution across textual units (*term-context model*). Such DSMs have become an indispensable ingredient in many NLP applications that require flexible broad-coverage lexical semantics.

Distributional modelling is an empirical science. DSM representations are determined by a wide range of parameters such as size and type of the co-occurrence context, feature selection, weighting of co-occurrence frequencies (often with statistical association measures), distance metric, dimensionality reduction method and the number of latent dimensions used. Despite recent efforts to carry out systematic evaluation studies, the precise effects of these parameters and their relevance for different application settings are still poorly understood.

The **wordspace** package aims to provide a flexible, powerful and easy to use "interactive laboratory" that enables its users to build DSMs and experiment with them, but that also scales up to the large models required by real-life applications.

Further background information and references can be found in:

> 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 110–114, Dublin, Ireland.

Before continuing with this tutorial, load the package with

```r
library(wordspace)
```

## Input formats

The most general representation of a distributional model takes the form of a sparse matrix, with entries specified as a triplet of row label (*target term*), column label (*feature term*) and co-occurrence frequency. A sample of such a table is included in the package under the name `DSM_VerbNounTriples_BNC`, listing syntactic verb-noun co-occurrences in the British National Corpus:

| | noun | rel | verb | f | mode |
|---|---|---|---|---|---|
| 338032 | exchange | subj | begin | 10 | written |
| 702570 | pattern | subj | break | 6 | written |
| 1238116 | casualty | obj | sustain | 5 | written |
| 1495496 | force | obj | feel | 45 | written |
| 1531186 | government | obj | assure | 7 | written |
| 1639598 | kitchen | obj | take | 5 | written |
| 1704102 | material | obj | produce | 171 | written |
| 1886882 | progress | obj | chart | 30 | written |
| 2064903 | status | obj | enhance | 21 | written |
| 2110292 | tea | obj | sell | 6 | written |

The `wordspace` package creates DSM objects from such triplet representations, which can easily be imported into R from a wide range of file and database formats. Ready-made import functions are provided for TAB-delimited text files (as used by DISSECT), which may be compressed to save disk space, and for term-document models created by the text-mining package `tm`.

The native input format is a pre-compiled sparse matrix representation generated by the UCS toolkit. In this way, UCS serves as a hub for the preparation of co-occurrence data, which can be collected from dependency pairs, extracted from a corpus indexed with the IMS Corpus Workbench or imported from various other formats.

## Creating a DSM

The first step in the creation of a distributional semantic model is the compilation of a co-occurrence matrix. Let us illustrate the procedure for verb-noun co-occurrences from the written part of the British National Corpus. First, we extract relevant rows from the table above.

```
Triples <- subset(DSM_VerbNounTriples_BNC, mode == "written")
```

Note that many verb-noun pairs such as *(walk, dog)* still have multiple entries in `Triples`: *dog* can appear either as the subject or as the object of *walk*.

```
subset(Triples, noun == "dog" & verb == "walk")
```

```
##         noun  rel verb  f    mode
## 295011   dog subj walk 20 written
## 1398669  dog  obj walk 87 written
```

There are two ways of dealing with such cases: we can either add up the frequency counts (a *dependency-filtered model*) or treat "dog-as-subject" and "dog-as-object" as two different terms (a *dependency-structured model*). We opt for a dependency-filtered model in this example – can you work out how to compile the corresponding dependency-structured DSM in R, either for verbs of for nouns as target terms?

The `dsm` constructor function expects three vectors of the same length, containing row label (target term), column label (feature term) and co-occurrence count (or pre-weighted score) for each nonzero cell of the co-occurrence matrix. In our example, we use nouns as targets and verbs as features. Note the option `raw.freq=TRUE` to indicate that the matrix contains raw frequency counts.

```
VObj <- dsm(target=Triples$noun, feature=Triples$verb, score=Triples$f, raw.freq=TRUE)
dim(VObj)
```

```
## [1] 10940  3149
```

The constructor automatically computes marginal frequencies for the target and feature terms by summing over rows and columns of the matrix respectively. The information is collected in data frames `VObj$rows` and `VObj$cols`, together with the number of nonzero elements in each row and column:

```
subset(VObj$rows, rank(-f) <= 6) # 6 most frequent nouns
```

```
##         term nnzero     f
## 5395     man    907 51976
## 6491  people    911 72832
## 6951 problem    402 29002
## 8928   thing    442 34104
## 8975    time    658 46500
## 9613     way    738 46088
```

This way of computing marginal frequencies is appropriate for syntactic co-occurrence and term-document models. In the case of surface co-occurrence based on token spans, the correct marginal frequencies have to be provided separately in the `rowinfo=` and `colinfo=` arguments (see `?dsm` for details).

The actual co-occurrence matrix is stored in `VObj$M`. Since it is too large to display on screen, we extract the top left corner with the `head` method for DSM objects. Note that you can also use `head(VObj, Inf)` to extract the full matrix.

```r
head(VObj)
```

```
## 6 x 6 sparse Matrix of class "dgCMatrix"
##              be have say give take achieve
## aa            7    5  12    .    .       .
## abandonment  14    .   .    .    .       .
## abbey        45   13   6    .    .       .
## abbot        23    7  10    5    5       .
## abbreviation  9    .   .    .    .       .
## abc           6    .   .    .    .       .
```

## The DSM parameters

Rows and columns with few nonzero cells provide unreliable semantic information and can lead to numerical problems (e.g. because a sparse association score deletes the remaining nonzero entries). It is therefore common to apply frequency thresholds both on rows and columns, here in the form of requiring at least 3 nonzero cells. The option `recursive=TRUE` guarantees that both criteria are satisfied by the final DSM when rows and columns are filtered at the same time (see the examples in `?subset.dsm` for an illustration).

```r
VObj <- subset(VObj, nnzero >= 3, nnzero >= 3, recursive=TRUE)
dim(VObj)
```

```
## [1] 6087 2139
```

If you want to filter *only* columns or rows, you can pass the constraint as a named argument: `subset=(nnzero >= 3)` for rows and `select=(nnzero >= 3)` for columns.

The next step is to weight co-occurrence frequency counts. Here, we use the *simple log-likelihood* association measure with an additional logarithmic transformation, which has shown good results in evaluation studies. The `wordspace` package computes *sparse* (or "positive") versions of all association measures by default, setting negative associations to zero. This guarantees that the sparseness of the co-occurrence matrix is preserved. We also normalize the weighted row vectors to unit Euclidean length (`normalize=TRUE`).

```r
VObj <- dsm.score(VObj, score="simple-ll", transform="log", normalize=TRUE, method="euclidean")
```

Printing a DSM object shows information about the dimensions of the co-occurrence matrix and whether it has already been scored. Note that the scored matrix does not replace the original co-occurrence counts, so `dsm.score` can be executed again at any time with different parameters.

```r
VObj
```

```
## Distributional Semantic Model with 6087 rows x 2139 columns
## * raw co-occurrence matrix M available
##   - sparse matrix with 191.4k / 13.0M nonzero entries (fill rate = 1.47%)
##   - in canonical format
##   - known to be non-negative
##   - sample size of underlying corpus: 5010.1k tokens
## * scored matrix S available
##   - sparse matrix with 153.7k / 13.0M nonzero entries (fill rate = 1.18%)
##   - in canonical format
##   - known to be non-negative
```

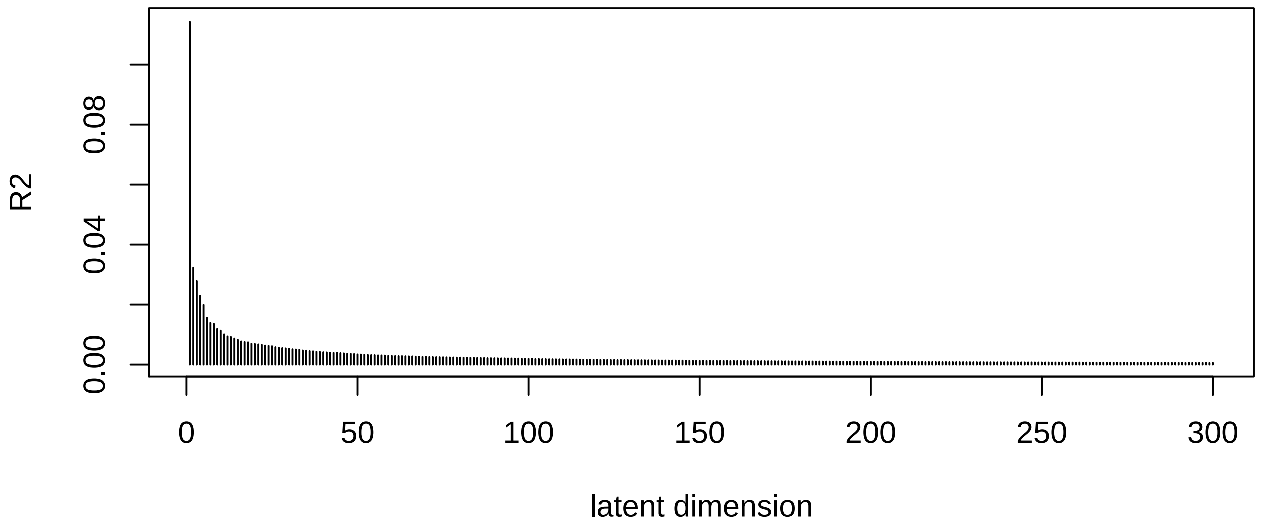
Most distributional models apply a dimensionality reduction technique to make data sets more manageable and to refine the semantic representations. A widely-used technique is singular value decomposition (SVD). Since `VObj` is a sparse matrix, `dsm.projection` automatically applies an efficient algorithm from the `sparsesvd` package.

```r
VObj300 <- dsm.projection(VObj, method="svd", n=300)
dim(VObj300)
```

```
## [1] 6087  300
```

`VObj300` is a dense matrix with 300 columns, giving the coordinates of the target terms in 300 latent dimensions. Its attribute `"R2"` shows what proportion of information from the original matrix is captured by each latent dimension.

```r
plot(attr(VObj300, "R2"), type="h", xlab="latent dimension", ylab="R2")
```



## Using DSM representations

The primary goal of a DSM is to determine "semantic" distances between pairs of words. The arguments to `pair.distances` can also be parallel vectors in order to compute distances for a large number of word pairs efficiently.

```
pair.distances("book", "paper", VObj300, method="cosine")
```

```
## book/paper
##   45.07627
```

By default, the function converts similarity measures into an equivalent distance metric – the angle between vectors in the case of cosine similarity. If you want the actual similarity values, specify `convert=FALSE`:

```
pair.distances("book", "paper", VObj300, method="cosine", convert=FALSE)
```

```
## book/paper
##  0.7061649
```

We are often interested in finding the nearest neighbours of a given term in the DSM space:

```
nearest.neighbours(VObj300, "book", n=14) # reduced space
```

```
##     paper   article      poem     works  magazine     novel      text
##  45.07627  51.92011  53.48027  53.91556  53.94824  54.40451  55.13910
##     guide newspaper  document      item     essay   leaflet    letter
##  55.27027  55.51492  55.62521  56.28246  56.29539  56.49145  58.04178
```

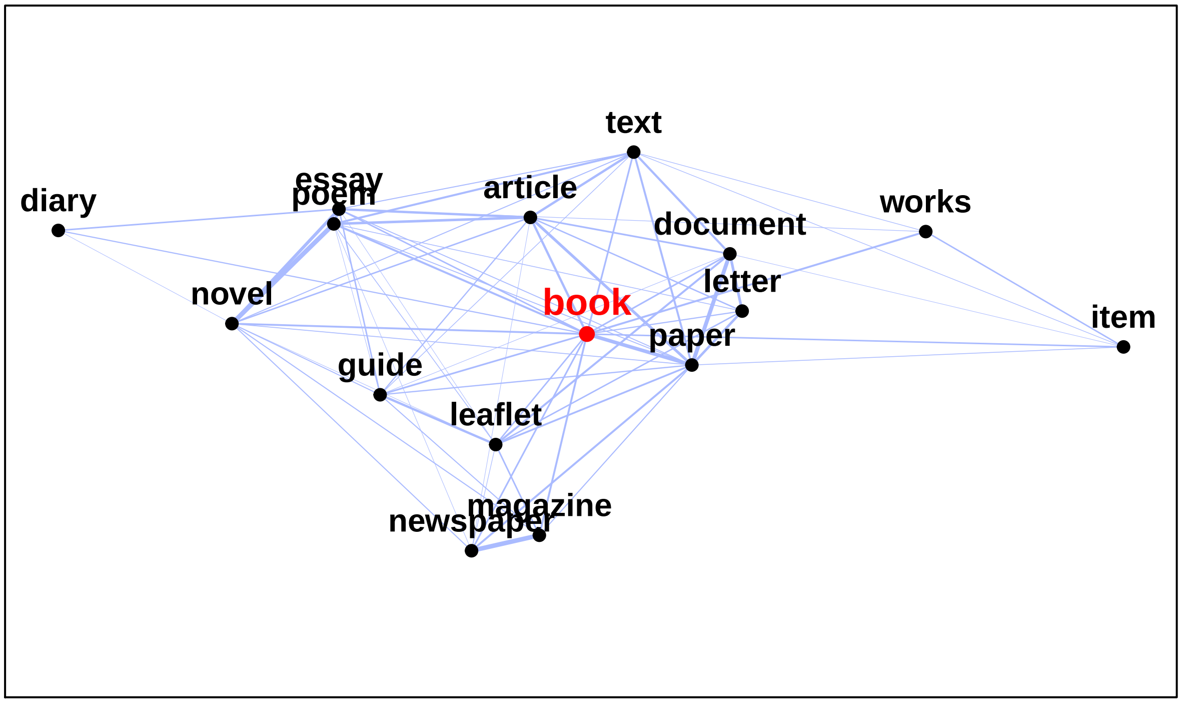
The return value is actually a vector of distances to the nearest neighbours, labelled with the corresponding terms. Here is how you obtain the actual neighbour terms:

```
nn <- nearest.neighbours(VObj, "book", n=15)    # unreduced space
names(nn)
```

```
##  [1] "paper"     "guide"     "works"     "novel"     "magazine"
##  [6] "article"   "document"  "poem"      "diary"     "essay"
## [11] "item"      "text"      "booklet"   "leaflet"   "newspaper"
```

The neighbourhood plot visualizes nearest neighbours as a semantic network based on their mutual distances. This often helps interpretation by grouping related neighbours. The network below shows that *book* as a text type is similar to *novel*, *essay*, *poem* and *article*; as a form of document it is similar to *paper*, *letter* and *document*; and as a publication it is similar to *leaflet*, *magazine* and *newspaper*.

```
nn.mat <- nearest.neighbours(VObj300, "book", n=15, dist.matrix=TRUE)
plot(nn.mat)
```



A straightforward way to evaluat distributional representations is to compare them with human judgements of the semantic similarity between word pairs. The `wordspace` package includes to well-known data sets of this type: Rubenstein-Goodenough (`RG65`) and `WordSim353` (a superset of `RG65` with judgements from new test subjects).

| | word1 | word2 | score |
|---|---|---|---|
| 5 | autograph_N | shore_N | 0.06 |
| 15 | monk_N | slave_N | 0.57 |
| 25 | forest_N | graveyard_N | 1.00 |
| 35 | cemetery_N | mound_N | 1.69 |
| 45 | brother_N | monk_N | 2.74 |
| 55 | autograph_N | signature_N | 3.59 |
| 65 | gem_N | jewel_N | 3.94 |

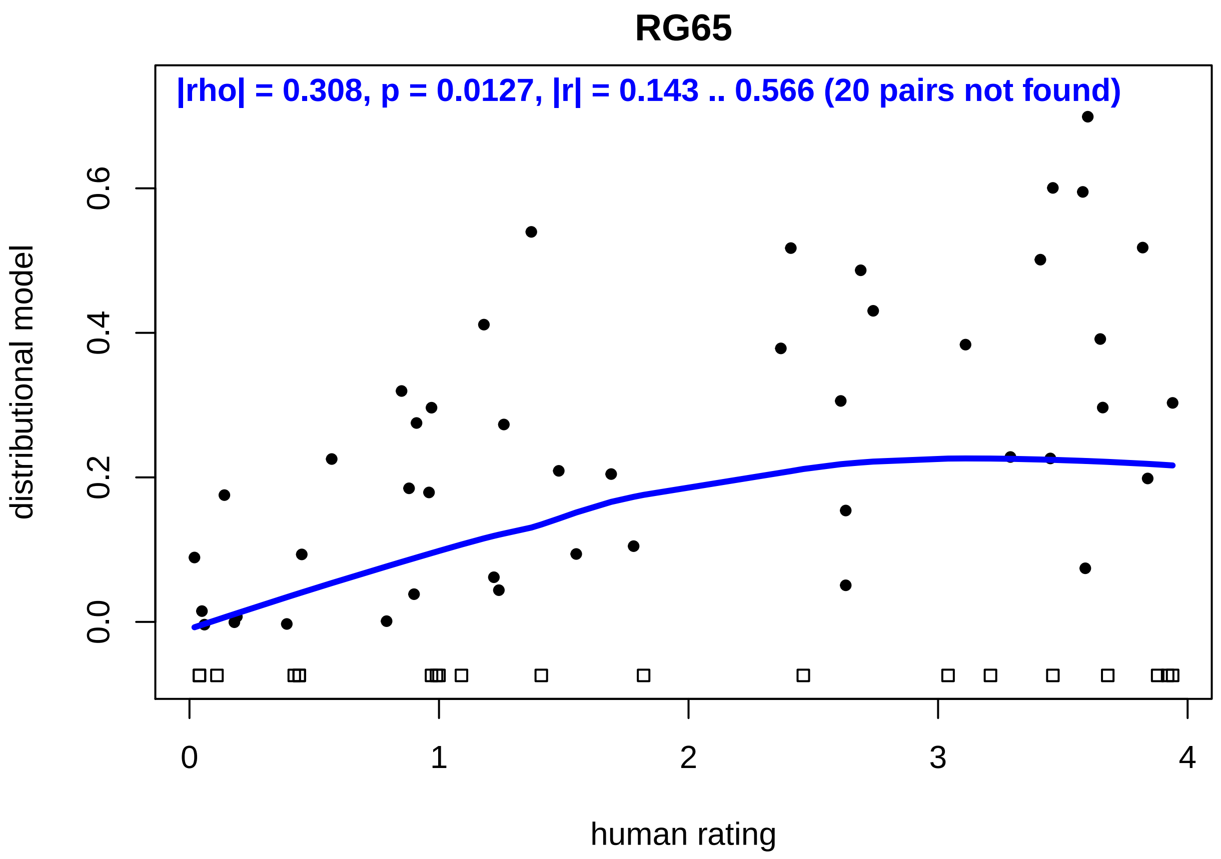
There is also a ready-made evaluation function, which computes Pearson and rank correlation between the DSM distances and human subjects. The option `format="HW"` adjusts the POS-disambiguated notation for terms in the data set (e.g. `book_N`) to the format used by our distributional model (`book`).

```r
eval.similarity.correlation(RG65, VObj300, convert=FALSE, format="HW")
```

```
##             rho    p.value missing         r   r.lower   r.upper
## RG65 0.3076154 0.01267694      20 0.3735435 0.1426399 0.5658865
```

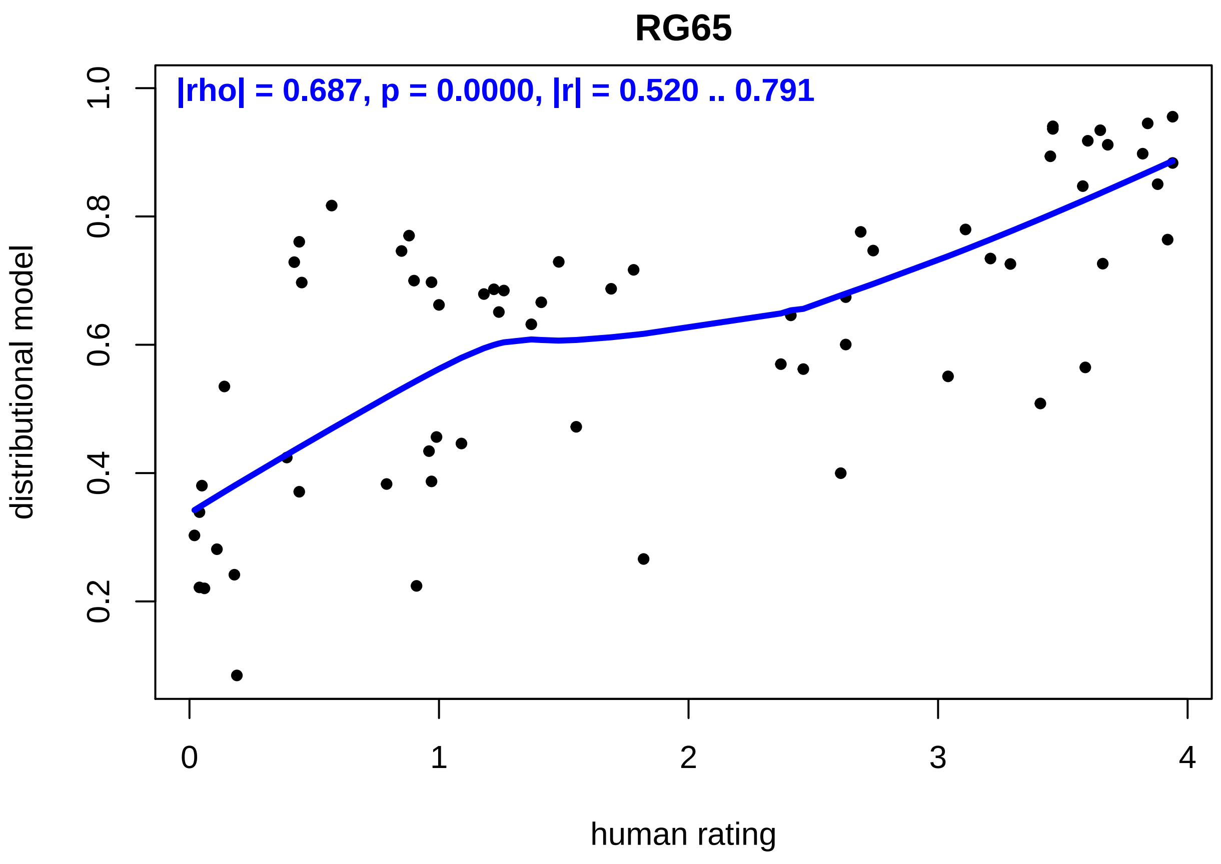
Evaluation results can also be visualized in the form of a scatterplot with a trend line.

```r
plot(eval.similarity.correlation(RG65, VObj300, convert=FALSE, format="HW", details=TRUE))
```



The rank correlation of 0.308 is very poor, mostly due to the small amount of data on which our DSM is based. Much better results are obtained with pre-compiled DSM vectors from large Web corpus, which are also included in the package. Note that target terms are given in a different format there (which corresponds to the format in `RG65`).

```
plot(eval.similarity.correlation(RG65, DSM_Vectors, convert=FALSE, details=TRUE))
```



## Advanced techniques

Schütze (1998) used DSM representations for word sense disambiguation (or, more precisely, word sense induction) based on a clustering of the sentence contexts of an ambiguous word. The `wordspace` package includes a small data set with such contexts for a selection of English words. Let us look at the noun *vessel* as an example, which has two main senses ("ship" and "blood vessel"):

```
Vessel <- subset(SemCorWSD, target == "vessel" & pos == "n")
table(Vessel$gloss)
```

```
##
## a craft designed for water transportation
##                                         6
##   a tube in which a body fluid circulates
##                                         6
```

Sentence contexts are given as tokenized strings (`$sentence`), in lemmatized form (`$hw`) and as lemmas annotated with part-of-speech codes (`$lemma`). Choose the version that matches the representation of target terms in your DSM.

| sense | sentence |
|---|---|
| vessel.n.02 | The spraying operation was conducted from the rear deck of a small Naval vessel , cruising two miles off-shore |
| vessel.n.01 | Such a dual derivation was strikingly demonstrated during the injection process where initial filling would be n |
| vessel.n.01 | This vessel could be followed to the parenchyma where it directly provided bronchial arterial blood to the alve |

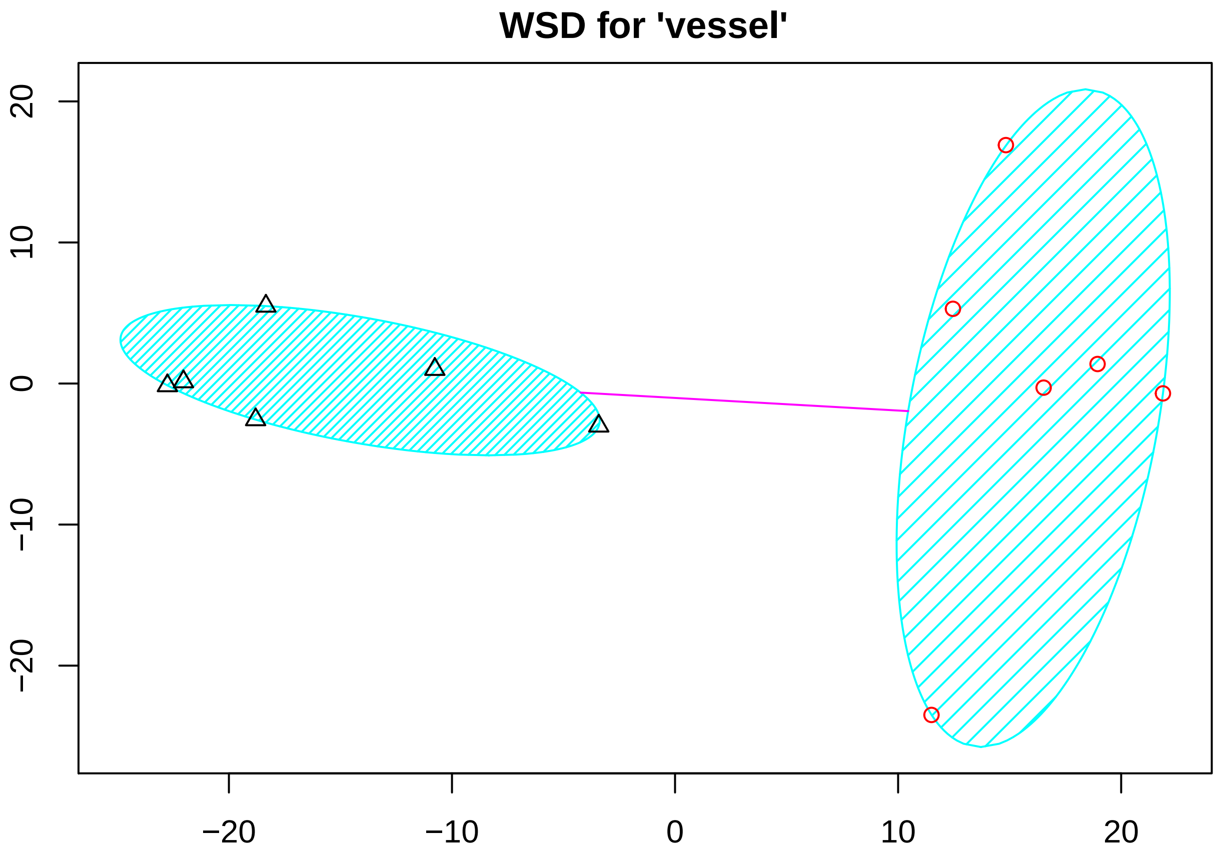
| sense | sentence |
|---|---|
| vessel.n.01 | However , this artery is known to be a nutrient vessel with a distribution primarily to the proximal airways an |
| vessel.n.01 | It is distinctly possible , therefore , that simultaneous pressures in all three vessels would have rendered the sh |
| vessel.n.01 | A careful search failed to show occlusion of any of the mesenteric vessels . |
| vessel.n.01 | Some of the small vessels were filled with fibrin thrombi , and there was extensive interstitial hemorrhage . |
| vessel.n.02 | He appeared to be peering haughtily down his nose at the crowded and unclean vessel that would carry him to |
| vessel.n.02 | However , we sent a third vessel out , a much smaller and faster one than the first two . |
| vessel.n.02 | Upon reaching the desired speed , the automatic equipment would cut off the drive , and the silent but not em |
| vessel.n.02 | Then , after slowing the vessel considerably , the drive would adjust to a one gee deceleration . |
| vessel.n.02 | To round out the blockading force , submarines would be needed - to locate , identify and track approaching v |

Following Schütze, each context is represented by a centroid vector obtained by averaging over the DSM vectors of all context words.

```
centroids <- context.vectors(DSM_Vectors, Vessel$lemma, row.names=Vessel$id)
```

This returns a matrix of centroid vectors for the 12 sentence contexts of *vessel* in the data set. The vectors can now be clustered and analyzed using standard R functions. Partitioning around medoids (PAM) has shown good and robust performance in evaluation studies.

```
library(cluster) # clustering algorithms of Kaufman & Rousseeuw (1990)
res <- pam(dist.matrix(centroids), 2, diss=TRUE, keep.diss=TRUE)
plot(res, col.p=factor(Vessel$sense), shade=TRUE, which=1, main="WSD for 'vessel'")
```



Colours in the plot above indicate the gold standard sense of each instance of *vessel*. A confusion matrix confirms perfect clustering of the two senses:

```
## table(res$clustering, Vessel$sense)
```

| vessel.n.01 | vessel.n.02 |
|---:|---:|
| 0 | 6 |
| 6 | 0 |

We can also use a pre-defined function for the evaluation of clustering tasks, which is convenient but does not produce a visualization of the clusters. Note that the "target terms" of the task must correspond to the row labels of the centroid matrix, which we have set to sentence IDs (`Vessel$id`) above.

```
eval.clustering(Vessel, M=centroids, word.name="id", class.name="sense")
```

```
##        purity entropy missing
## Vessel    100       0       0
```

As a final example, let us look at a simple approach to compositional distributional semantics, which computes the compositional meaning of two words as the element-wise sum or product of their DSM vectors.

```
mouse <- VObj300["mouse", ] # extract row vectors from matrix
computer <- VObj300["computer", ]
```

The nearest neighbours of mouse are problematic, presumably because the type vector represents a mixture of the two senses that is not close to either meaning in the semantic space.

```
nearest.neighbours(VObj300, "mouse", n=12)
```

```
##        prop     isotope      carbon       serum transformer      sponge
##    53.72837    55.08473    55.29022    56.84607    57.20004    57.25144
## thermometer        hoop loudspeaker       razor       mount     implant
##    57.33371    57.38682    57.38682    57.43123    57.47889    57.49171
```

By adding the vectors of *mouse* and *computer*, we obtain neighbours that seem to fit the "computer mouse" sense very well:

```
nearest.neighbours(VObj300, mouse + computer, n=12)
```

```
##      mouse   computer    program  processor   software       tool
##   36.69755   41.47028   54.15183   54.37412   54.55897   55.07898
##    machine transistor        mix   keyboard       prop     sponge
##   55.74764   58.51586   58.93807   59.15682   59.18378   59.34185
```

Note that the target is specified as a distributional vector rather than a term in this case. Observations from the recent literature suggest that element-wise multiplication is not compatible with non-sparse SVD-reduced DSMs, so it is not surprising to find completely unrelated nearest neighbours in our example:

```
nearest.neighbours(VObj300, mouse * computer, n=12)
```

```
##  picasso    stein    clamp    ivory   copper    crane   carter    bruce
## 44.14916 44.84104 45.81757 46.05523 48.03508 48.82008 51.12203 51.38382
## amaranth fielding   bowman  griffin
## 51.56460 51.60627 51.62372 51.62551
```