CHAPTER 16

Introduction to Semantic Vector Spaces

Cosine as a measure of semantic similarity

What you will learn from this chapter:

This chapter introduces Semantic Vector Spaces, another distributional approach to semantics. This method originates in Natural Language Processing. Unlike Behavioural Profiles discussed in the previous chapter, it uses automatically extracted co-occurrences of target words and contextual features. The characteristic features of the method are weighted co-occurrence frequencies and the use of the cosine as the most popular similarity measure. This chapter provides a general introduction to the method, with a case study of English cooking verbs as an illustration.

16.1 Distributional models of semantics and Semantic Vector Space models

The main idea behind the distributional approach was formulated by John Firth in his famous quotation, "You shall know a word by the company it keeps" (Firth 1957). To put it differently, words that occur in the same contexts tend to have similar meanings (Harris 1954). This idea lies at the core of Semantic Vector Space modelling. Originating in Computational Linguistics (e.g. Deerwester et al. 1990; Schütze 1992; Lin 1998), this approach is now finding its way into lexical semantics and other theoretical disciplines. Similar to Behavioural Profiles, Semantic Vector Spaces are based on comparing contextual environments of words or senses, but in a more computationally intensive way. Instead of a manually coded sample, researchers extract the distributional information automatically from large morphologically and syntactically annotated corpora. The weighted co-occurrence frequencies between target words and contextual features are combined in long vectors. One can compute the distances between the vectors in order to measure their semantic similarity, hence the term 'Vector Spaces'.

Implementations of Semantic Vector Spaces vary with regard to how one defines the context: as entire documents, 'bags of words' that contain a certain number of lexemes on the right and left from the target word, syntactic subcategorization frames, etc. The choice of the context type is crucial. If the context is defined as a bag of words, it will be particularly sensitive to topic-related contextual differences and semantic prosody. An example

is Peirsman et al. (2010), who demonstrate the change in lexical associations of the word *islam* after the terrorist attacks of 9/11 in Dutch newspapers. However, the size of the context window around the target word also matters: smaller context windows help model semantic relationships, such as synonymy, hyper- and hyponymy, cohyponymy, whereas wider context windows are more useful in modelling topical similarity.

If the definition of context also takes into account the syntactic relationships between a word and its collocates, such models are better at capturing semantic similarity (Pado & Lapata 2007; Peirsman 2008). If the context is defined on the basis of purely syntactic features, e.g. subcategorization frames, the words tend to cluster according to more abstract grammatical distinctions, e.g. Levin-like verb classes and broad lexicosemantic noun classes in Levshina & Heylen (2014). In this case study we will explore a basic bag-of-words model. However, the principles of analysis are not fundamentally different for other definitions of contexts.

Why do contextual features work as a proxy for semantics? Some linguists and philosophers, including Wittgenstein, go as far as to suggest that meaning *is* context of use (see Stefanowitsch 2010: 368–370). A less radical interpretation of the role of contextual clues is that the latter are related to the conceptual structures associated with linguistic forms. It has been shown that the situational clues present in the context of a new word or construction are crucial in establishing their meaning, especially when learning non-basic vocabulary, which normally happens without immediate access to referents (Dąbrowska 2009). In psycholinguistics, Miller & Charles (1991) found experimental evidence that similarity judgments about words depend on the contexts in which the words appear. Some distributional models applied in psycholinguistic research, such as Latent Semantic Analysis (Landauer & Dumais 1997), have been quite successful in performing human lexical tasks, for example, a TOEFL test on synonyms.

16.2 A Semantic Vector Space model of English verbs of cooking

16.2.1 Theoretical background and data

For this case study you will need the data and functions from the two add-on packages. They should first be installed if you have not installed them yet. Next, you should load them.

```
> install.packages("cluster")
> library(Rling); library(cluster)
```

Semantic (conceptual) and lexical fields were a central topic in structuralist lexicology and anthropology. Every word in a lexical field was believed to acquire its meaning by its opposition to the other words in the same field. Some words can be more general (hypernyms) and some more specific (hyponyms). For instance, consider English verbs

of cooking.[1] Figure 16.1 represents some of the verbs organized in a hierarchical network (an adapted version of analysis in Lehrer 1974). The verb *cook* is the superordinate term, which designates an irreversible process involving heat with a purpose of making food more desirable and/or nutritious and/or digestible (Lehrer 1974: 62). The other verbs inherit these properties. Figure 16.1 shows the most important semantic features that distinguish between the verbs, for instance *+water* or *-fat* (oil). The structure of the semantic field is quite complex. There are two kinds of *boil*: one of them is a superordinate term that designates cooking with water and without fat, and the other one is the former's hyponym with the feature *+vigorous* (*cooking*). This kind of semantic relationships is called autohyponymy. Some terms may have more than two superordinates: *grill* can be applied both to cooking food on an open grill, as a subordinate term of *broil*, and to the process of cooking with fat on a frying pan, as a subordinate term of *fry* (e.g. *grilled cheese sandwich*). The position of *roast* as a hyponym of both *bake* and *broil* can be explained by the changes in the cooking practices. Previously, foods used to be roasted on a spit in front of an open fire; traditionally, *roast* is similar to *broil* and its subordinate terms. However, nowadays the same effect can be achieved by putting food in an oven, similar to baking.

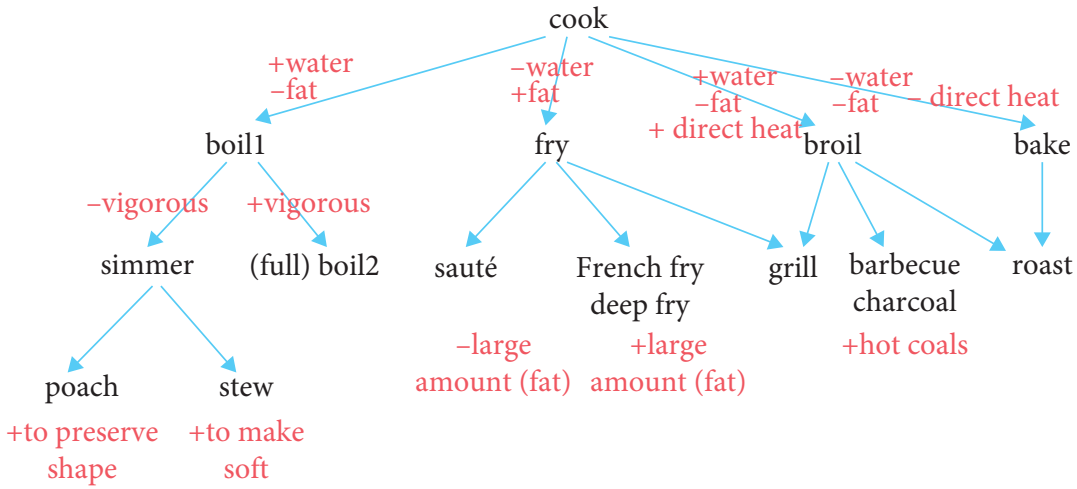

Figure 16.1. English verbs of cooking (a selection), an adjusted version of Lehrer (1974)

Although the abstract distinctive features play a crucial role, Lehrer observes that the lower the verb in the hierarchy, the greater role is played by its collocates (Lehrer 1974: 33). For instance, *poach* means that the food is carefully boiled in water so that the shape is preserved. A typical example is *poached eggs*. *Stew* means that the food is cooked slowly for a long time until it is soft, e.g. *stewed vegetables*. However, the phrase ?*poached vegetables* sounds strange, even if the vegetables are cooked so that their shape is preserved. On the

1. The graph represents the American English terms (around 1970s); in British English *grill* is used instead of *broil*.

other hand, ?*stewed eggs* is not fully acceptable, either, even if the eggs are cooked slowly. Another problem is the relationships between the subordinate and superordinate terms, which are not always transitive. For instance, *poached egg* is not equal to *boiled egg*. In the former case, unlike in the latter, the shell is removed before cooking.

The hypothesis of this case study is that collocates are important at all levels and that we can obtain a similar clustering of cooking verbs by only looking at their distributional properties. For the case study, a sample of ten verbs was selected: *bake*, *boil*, *broil*, *cook*, *fry*, *grill*, *poach*, *roast*, *simmer* and *stew*. All instances of these verbs were collected from the Corpus of Contemporary American English (Davies 2008 –). To ensure maximally that the verbs were used in the literal sense, the data were selected from magazines (where most uses come from recipes) and the verbs were used only in the initial [vv0] form, which overlaps with the imperative (e.g. *bake for 40 to 50 minutes*). For a bag-of-words vector space model, collocates (lemmas) were selected within the window of five words on the left and five words on the right. Next, a matrix of co-occurrences was created, which can be found in the data frame `cooking`:

```
> data(cooking)
> head(cooking)
            Bake Boil Broil Cook Fry Grill Poach Roast Stew Simmer
AB             2    0     0    0   0     0     0     0    0      0
ABOUTI         0    0     0    2   0     0     0     0    0      0
ABRAMS         0    0     0    0   0     1     0     0    0      0
ABSORB         0    0     0    0   0     0     0     0    0     14
ABSORBED-10    0    0     0    1   0     0     0     0    0      0
ABUNDANCE      0    0     0    0   1     0     0     0    0      0
```

The analysis is done in three steps:

Step 1. Creation of semantic vectors from weighted co-occurrence frequencies:
- a) computation of expected co-occurrence frequencies;
- b) computation of Pointwise Mutual Information scores;
- c) transformation of the latter into Positive Pointwise Mutual Information (PPMI) scores.

Step 2. Computation of similarity scores between the resulting PPMI vectors with the help of the cosine measure.

Step 3. Exploration of similarity scores – if necessary, with the help of cluster analysis.

16.2.2 Creating vectors of weighted co-occurrence frequencies

To begin with the analyses, we will first remove from the data frame `cooking` all rows that contain only zeros. These are collexemes that do not occur with any of the target verbs we are interested in.

```
> cooking <- cooking[rowSums(cooking) > 0, ]
```

The dataset contains 'raw' co-occurrence frequencies. It is common to weight them in order to give more prominence to surprising co-occurrences. The most popular weighting method in Semantic Vector Spaces is Pointwise Mutual Information scores. It is highly sensitive to combinations of less frequent targets and features. The formula of PMI of target word x and contextual feature y is presented below. Note that it is common to use the logarithm to base 2:

$$\mathrm{PMI}(x, y) = \log_2 \frac{p(x, y)}{p(x)p(y)}$$

where $p(x, y)$ is the probability of co-occurrence of x and y, $p(x)$ is the probability of x, and $p(y)$ is the probability of y. This formula can be expressed in a more convenient form (cf. Chapter 10):

$$\mathrm{PMI}(x, y) = \log_2 \frac{O_{xy}}{E_{xy}}$$

where O_{xy} is the observed frequency of co-occurrence of x and y, and E_{xy} is the expected frequency.

The notions of observed and expected frequencies were introduced in Chapter 9. To obtain a vector of expected frequencies for a verb and all its collocates, you can use the following code:

```
> exp.bake <- sum(cooking$Bake)*rowSums(cooking)/sum(cooking)
> exp.bake[1:5]
AB          ABOUTI      ABRAMS      ABSORB      ABSORBED-10
0.5737930   0.5737930   0.2868965   4.0165510   0.2868965
```

The result is a vector of expected frequencies for *bake* and each collocate. Now we can create a vector of PMI scores. Again, below is the code for *bake*:

```
> PMI.bake <- log2(cooking$Bake/exp.bake)
> PMI.bake[1:5]
AB         ABOUTI  ABRAMS   ABSORB   ABSORBED-10
1.801398   -Inf    -Inf     -Inf     -Inf
```

It has been shown in Bullinaria & Levy (2007) that the Positive Pointwise Mutual Information (PPMI) measure performs better than simple PMI scores. PPMI is essentially the same as PMI, but all negative values are replaced with zeros. Below is the code for *bake*:

```
> PPMI.bake <- ifelse(PMI.bake < 0, 0, PMI.bake)
> PPMI.bake[1:5]
AB         ABOUTI    ABRAMS    ABSORB    ABSORBED-10
1.801398   0.000000  0.000000  0.000000  0.000000
```

To apply this procedure to all verbs simultaneously, one can use the code below. First, we will turn the data frame into a matrix to facilitate further calculations.

```
> cooking <- as.matrix(cooking)
```

Next, we will compute expected frequencies for all cells in the data by using `chisq.test()`, which was introduced in Chapter 9:

```
> cooking.exp <- chisq.test(cooking)$expected
[warning message omitted]
```

After that, we compute the PMI and PPMI scores for all verbs and all collocates:

```
> cooking.PMI <- log2(cooking/cooking.exp)
> cooking.PPMI <- ifelse(cooking.PMI < 0, 0, cooking.PMI)
```

Now we are ready for the next step, computation of the cosine similarity values.

16.2.3 Cosine similarity

Now we need to compute the distances between the vectors. One option is to use the `dist()` function, as in the previous case study. Yet, it is more conventional to use the cosines of angles between distributional vectors as measures of (dis)similarity. Consider Figure 16.2. Angle θ_2 between the imaginary distributional vectors of the nouns *dog* and *cat* is smaller than the angle θ_1 between *dog* and *Universe*, since *dog* and *cat* are more semantically related and therefore should occur in similar contexts. For the angle between 0^0 and 180^0, the greater the angle, the smaller the cosine of that angle. If the angle between two PPMI vectors is 0^0, which means maximal distributional similarity, the cosine value is 1. If the angle between two PPMI vectors is 90^0, which indicates maximal dissimilarity, the cosine value is 0. So, the greater the similarity of two vectors, the greater the cosine value of the angle between them. In this example, the cosine similarity of *dog* and *cat* will be greater than that of *dog* and *Universe*.

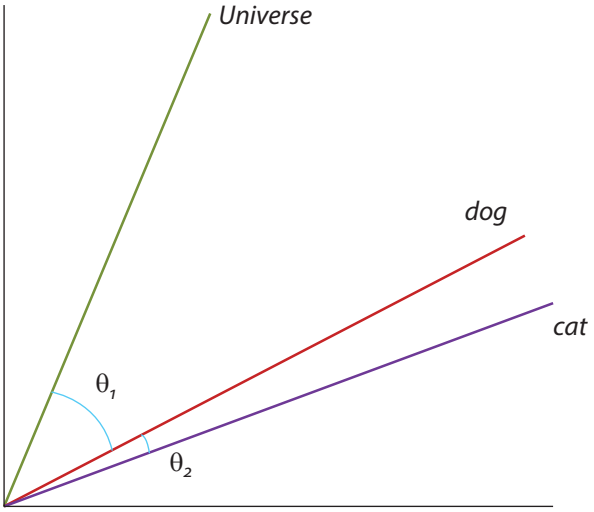

Figure 16.2. Cosine as a similarity measure

The code to compute the cosine between two vectors *a* and *b* can be represented as follows:

```
> cos <- sum(a*b)/(sqrt(sum(a *a)) *sqrt(sum(b *b))) # do not run
```

or, alternatively,

```
> cos <- crossprod(a, b)/sqrt(crossprod(a)*crossprod(b)) # do not run
```

For instance, the cosine between the distributional vectors of *bake* and *simmer* is as follows:

```
> crossprod(cooking.PPMI[, 1], cooking.PPMI[, 10])/
sqrt(crossprod(cooking.PPMI[, 1]) *crossprod(cooking.PPMI[, 10]))
           [,1]
[1,] 0.01160899
```

The cosine value for *boil* and *simmer* is greater, which means that the verbs are more similar distributionally:

```
> crossprod(cooking.PPMI[, 2], cooking.PPMI[, 10])/
sqrt(crossprod(cooking.PPMI[, 2]) *crossprod(cooking.PPMI[, 10]))

     [,1]
[1,] 0.07422152
```

It would be too time-consuming to compute all cosines pairwise. To speed up the process, one can use the function `cossim()` in the `Rling` package. Its argument is a matrix with distributional vectors as rows, so we will first transpose the matrix with PPMI scores:

```
> cooking1 <- t(cooking.PPMI)
> cooking.cos <- cossim(cooking1)
> round(cooking.cos, 2)
       Bake Boil Broil Cook Fry  Grill Poach Roast  Stew  Simmer
Bake   1.00 0.01 0.06  0.01 0.03 0.01  0.01  0.04   0.00  0.01
Boil   0.01 1.00 0.03  0.03 0.03 0.02  0.03  0.05   0.03  0.07
Broil  0.06 0.03 1.00  0.03 0.04 0.08  0.03  0.08   0.03  0.02
Cook   0.01 0.03 0.03  1.00 0.03 0.03  0.01  0.03   0.02  0.03
Fry    0.03 0.03 0.04  0.03 1.00 0.06  0.03  0.05   0.01  0.01
Grill  0.01 0.02 0.08  0.03 0.06 1.00  0.04  0.07   0.03  0.01
Poach  0.01 0.03 0.03  0.01 0.03 0.04  1.00  0.03   0.07  0.04
Roast  0.04 0.05 0.08  0.03 0.05 0.07  0.03  1.00   0.02  0.03
Stew   0.00 0.03 0.03  0.02 0.01 0.03  0.07  0.02   1.00  0.02
Simmer 0.01 0.07 0.02  0.03 0.01 0.01  0.04  0.03   0.02  1.00
```

This is a similarity matrix, the opposite of a distance matrix, which was discussed in Chapter 15. The diagonal elements are equal to 1 because a vector is maximally similar to itself. In contrast, the diagonal elements in a distance matrix are always zeros. If we examine the cosine values row by row or column by column, we can already detect some interpretable interesting patterns:

– *broil*, *roast* and *grill* have the highest similarity scores;
– *simmer* and *boil* are relatively similar to each other;
– *poach* and *stew* are relatively similar to each other.

It is also interesting that the superordinate *cook* is not particularly close to any other verb, having medium-range similarity scores (around 0.03) with most of them.

The cosine similarity measures can be transformed into distances. The easiest method would be simply to subtract each similarity score from 1. Here we will use another method. We will divide each similarity score by the maximum similarity (excluding one) in the data and subtract the result from one. This operation will produce normalized distance values more evenly distributed in a range from 0 to 1. The object can next be transformed into a full-fledged distance matrix with the help of `as.dist()`:

```
> cooking.dist <- 1 - (cooking.cos/0.07958)
> cooking.dist <- as.dist(cooking.dist)
```

Now we are ready to perform clustering with the help of partitioning around medoids, which was introduced in Chapter 15. When using a partitioning clustering method, one should specify in advance the number of clusters into which the data will be classified. Since we do not know the optimal number of clusters in advance, we will try different solutions, from two to nine, and compare their average silhouette widths. As was discussed in Chapter 15, the average silhouette width is a convenient tool for estimating the optimal number of clusters. We will use the function `pam()` in the package `cluster`. Below you can see the code that can be used to compute the average silhouette width of a two-cluster solution.

```
> test.clust <- pam(cooking.dist, 2)
> test.clust$silinfo$avg.width
[1] 0.3664788
```

If we repeat the procedure for the number of clusters up to 9, we will see that the best solution seems to be the one with six clusters because it yields the highest average silhouette width 0.6. See the previous chapter (Section 15.2.5.3) on how to write one line of code to compute average silhouette width based on PAM-clustering.

Let us look at the six-cluster solution more closely.

```
> test.clust <- pam(cooking.dist, 6)
> test.clust$clustering
Bake  Boil  Broil  Cook  Fry  Grill  Poach  Roast  Stew  Simmer
1     2     3      4     5    3      6      3      6     2
```

The largest cluster 3 includes three verbs: *broil*, *grill* and *roast*. Cluster 2 contains *boil* and *simmer*. Cluster 6 includes *poach* and *stew*. The other verbs (*bake*, *cook* and *fry*) form their own individual clusters.

The results are very close to Lehrer's classification. We have found that the co-hyponyms *poach* and *stew* cluster together, as well as *simmer* and *boil*. *Broil* clusters with

its co-hyponyms, *grill* and *roast*. The verb *bake* is on its own, since it has unique properties, and so is *fry*. *Cook*, as the most general term, is on its own, as well. Note that Lehrer's hierarchy contained quite a few ambiguities due to polysemy, and further research is needed to take these nuances into account, for instance, by creating distributional vectors for each sense, rather than for a lexeme.

16.3 Summary

In this chapter, we have introduced the method of Semantic Vector Spaces. Unlike the Behavioural Profiles method, which involves individual corpus instances of constructions coded for categorical variables, normally selected on theoretical grounds, Semantic Vector Spaces are based on automatically collected co-occurrence frequencies of target words with many other words and/or syntactic relationships or frames, and represent a radically data-driven technique. In spite of all differences, there are two major assumptions shared by both Behavioural Profiles and Semantic Vector Spaces: first, distributional differences are interpreted as semantic differences, and second, these differences are treated as spatial distances. These distances were measured as cosines of angles between distributional vectors in semantic hyperspace.

How to report analyses based on Semantic Vector Spaces

There are no strict rules, and the procedure strongly depends on the specific task. However, one should mention the character and number of contextual features, the weighting method, and the similarity/distance measure. Usually the performance of a model is evaluated by comparing it with a 'gold standard', such as WordNet or results of psycholinguistic experiments.

In practice, Semantic Vector Spaces are created on much longer lists of target words, and contextual features are often carefully pre-selected (e.g. a thousand most frequent nouns). One also uses computationally efficient, fast algorithms written in other programming languages, such as Python or Java. However, the main aim of this chapter was to introduce the basic principles of Semantic Vector Spaces and demonstrate how semantics can be done in a data-driven, bottom-up fashion.

More on Semantic Vector Spaces

The literature on vector spaces in linguistics is abundant. For a linguist-friendly introduction, see Levshina & Heylen (2014). For more technical details and a broader picture, see Turney & Pantel (2010). An important application concerns compositional semantics (e.g. Mitchell & Lapata 2010). See also a discussion of distributional Latent Semantic Analysis in a pioneering paper by Landauer & Dumais (1997).