In the former chapter, we made clear that the context of a particular word is able to suitably inform us about its semantics, and we looked at the various contexts that might be useful for the induction of semantic similarity. In this chapter, we will investigate how this notion of context can be formally implemented in a computational framework.

2.1 Formal model

The former chapter provides an intuitive idea of how the context of a word might be used to calculate its semantic similarity to other words. Now, in order to implement this idea in a computational framework, it needs to be expressed in more formal terms. In this section, we will have a look at some existing literature that stipulates semantic space models and the notion of context in more formal terms.

Lowe (2001) provides a formal definition of a semantic space model; he defines the model as a quadruple \( (A, B, S, M) \). \( B \) is a set of basic elements \( (b_1 \ldots b_D) \) determining the dimensionality \( D \) of the vector space and the interpretation of each dimension. \( B \) might be a set of documents, words, or dependency relations, depending on the context that is used. \( A \) specifies the function that maps the standard co-occurrence frequencies of basis elements and words to their final value, so that each word is represented by a vector \( v = [A(b_1, t), A(b_2, t), \ldots, A(b_D, t)] \). \( A \) may be the identity function (so that the final vector contains simple co-occurrence counts), but often a more advanced mapping is used. \( A \) is called the lexical
2.2 Similarity calculations: geometry vs. probability

Semantic similarity can be implemented in two different – albeit related – ways: in a (geometrically oriented) vector space model or in a (statistically oriented) probability distribution model. Both models are instantiations of the formal model described above. We will discuss the former in section 2.2.1 and the latter in section 2.2.2. Different similarity measures $S$ are presented for both models.
2.2.1 Vector space model

In a semantic vector space model, each word in a language is mapped to a point in a real finite dimensional vector space. The vector space model is one of the most widely used models for the acquisition of semantic similarity. The model makes it possible to express ‘semantic proximity’ between entities in terms of spatial distance. In a vector space model, particular entities (words, for example) are represented as vectors of features (the word’s different contexts) in a multi-dimensional Euclidean space. By applying a suitable similarity measure (cfr. infra), one can straightforwardly calculate the similarity between the different entities.

The vector space model was first developed in the context of information retrieval (Salton, Wong, and Yang, 1975), representing documents and queries as vectors of the words they contain. The documents that are the closest to a particular query in this vector space (i.e. the documents that are using the same words as in the query) will most likely represent the documents that the user was looking for.

This model can straightforwardly be applied to similarity calculations between words. The two words for which the semantic similarity is to be calculated, are represented as vectors of the words’ various contexts. Figure 2.1 shows an example matrix $M$ containing vectors for four different target words (using dependency relations as features). In this example the set of target words is

$$ T = \{\text{apple, banana, car, truck}\} $$

and the set of basic elements is

$$ B = \{\text{red}_{\text{adj}}, \text{yellow}_{\text{adj}}, \text{tasty}_{\text{adj}}, \text{fast}_{\text{adj}}, \text{eat}_{\text{obj}}, \text{drive}_{\text{obj}}\} $$

<table>
<thead>
<tr>
<th></th>
<th>red$_{\text{adj}}$</th>
<th>yellow$_{\text{adj}}$</th>
<th>tasty$_{\text{adj}}$</th>
<th>fast$_{\text{adj}}$</th>
<th>eat$_{\text{obj}}$</th>
<th>drive$_{\text{obj}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>apple</td>
<td>200</td>
<td>24</td>
<td>129</td>
<td>0</td>
<td>289</td>
<td>0</td>
</tr>
<tr>
<td>banana</td>
<td>1</td>
<td>152</td>
<td>87</td>
<td>1</td>
<td>214</td>
<td>1</td>
</tr>
<tr>
<td>car</td>
<td>120</td>
<td>74</td>
<td>0</td>
<td>98</td>
<td>1</td>
<td>386</td>
</tr>
<tr>
<td>truck</td>
<td>67</td>
<td>44</td>
<td>0</td>
<td>37</td>
<td>0</td>
<td>175</td>
</tr>
</tbody>
</table>

Figure 2.1: A noun-by-features matrix

The value in matrix cell $(i, j)$ is the co-occurrence frequency of word $i$ with value $j$. In the example above, the adjective red appears 200 times with the word apple, and the word car appears 386 times as the object of drive.
To facilitate computations, vectors are often normalized to vector length of 1. The vector length or norm of a vector $\overrightarrow{v}$ with length $k$ is calculated with equation 2.1.

$$ |\overrightarrow{v}| = \sqrt{\sum_{i=1}^{k} v_i^2 }$$  \hspace{1cm} (2.1)

Dividing a vector by its vector length normalizes it to a vector length of 1. When normalizing the vectors in figure 2.1, the resulting matrix looks like the one in figure 2.2.

<table>
<thead>
<tr>
<th></th>
<th>red$_{adj}$</th>
<th>yellow$_{adj}$</th>
<th>tasty$_{adj}$</th>
<th>fast$_{adj}$</th>
<th>eat$_{obj}$</th>
<th>drive$_{obj}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>apple</td>
<td>.533</td>
<td>.064</td>
<td>.344</td>
<td>.000</td>
<td>.770</td>
<td>.000</td>
</tr>
<tr>
<td>banana</td>
<td>.004</td>
<td>.550</td>
<td>.315</td>
<td>.004</td>
<td>.774</td>
<td>.004</td>
</tr>
<tr>
<td>car</td>
<td>.284</td>
<td>.175</td>
<td>.000</td>
<td>.232</td>
<td>.002</td>
<td>.914</td>
</tr>
<tr>
<td>truck</td>
<td>.342</td>
<td>.224</td>
<td>.000</td>
<td>.189</td>
<td>.000</td>
<td>.893</td>
</tr>
</tbody>
</table>

Figure 2.2: A noun-by-features matrix normalized by vector length

In order to calculate the contextual overlap between two vectors $\overrightarrow{v}$ and $\overrightarrow{w}$ (which – as has been described in the previous chapter – we think of as a good predictor for semantic similarity), we need a proper vector similarity measure $S = \text{sim}(\overrightarrow{v}, \overrightarrow{w})$.

The two simplest measures for vector similarity are the Manhattan distance and the Euclidean distance. The Manhattan distance or $L_1$ norm is defined as

$$ \text{dist}_{\text{MANHATTAN}}(\overrightarrow{v}, \overrightarrow{w}) = \sum_{i=1}^{k} |v_i - w_i|$$  \hspace{1cm} (2.2)

and the Euclidean distance, or $L_2$ norm, is defined as

$$ \text{dist}_{\text{EUCLIDEAN}}(\overrightarrow{v}, \overrightarrow{w}) = \sqrt{\sum_{i=1}^{k} (v_i - w_i)^2}$$  \hspace{1cm} (2.3)

Both distance measures are intuitively easy to understand, and provide a sound and straightforward extension of semantic similarity calculations in terms of spatial distance. In practice, though, neither the Manhattan distance nor the Euclidean distance are frequently used as word similarity measures.
Two other similarity measures that do show up in word similarity calculations – Jaccard and Dice – are derived from set theory. Originally, they were designed for binary vectors, but they can easily be extended in order to deal with frequency data.

The Jaccard similarity measure is defined as

$$sim_{\text{Jaccard}}(\vec{v}, \vec{w}) = \frac{\sum_{i=1}^{k} \min(v_i, w_i)}{\sum_{i=1}^{k} \max(v_i, w_i)}$$

(2.4)

and the Dice similarity measure is defined as

$$sim_{\text{Dice}}(\vec{v}, \vec{w}) = \frac{2 \times \sum_{i=1}^{k} \min(v_i, w_i)}{\sum_{i=1}^{k} (v_i + w_i)}$$

(2.5)

Intuitively, both measures calculate the weight of overlapping features (the numerator with the min function) compared to the total feature weight (the denominator, either using the max function for the Jaccard measure or the sum of both vectors’ feature values for the Dice measure).

The best known and most widely used similarity measure, however, is the cosine similarity measure. The cosine similarity is easy to compute, and it often achieves the best results. It has therefore become the best known and most widely used vector space similarity measure. The cosine similarity measure is calculated as

$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|}$$

(2.6)

where $\vec{v} \cdot \vec{w}$ is the dot product between vector $\vec{v}$ and $\vec{w}$, both of length $k$

$$\vec{v} \cdot \vec{w} = \sum_{i=1}^{k} v_i w_i$$

(2.7)

Note that, when both vectors $\vec{v}$ and $\vec{w}$ are normalized to unit length, the denominator is redundant, so that the cosine similarity amounts to a simple dot product between two vectors.

Once we have defined the similarity measure $S$, we can calculate the similarity between the different word vectors. The resulting calculation yields the similarity matrix $S_{sim}$ of size $n \times n$, where $n$ is the number of target words $T$. The similarity matrix is represented in figure 2.3.
2.2.2 Probabilistic model

The vector space model is the oldest, best known and most widely used model for semantic similarity, but it is not the only one. A word’s contextual information can also be captured in a statistically oriented probability distribution model. Probability distribution models allow for the use of well-known information-theoretic measures of similarity, and they offer the possibility of implementing semantic similarity in a Bayesian framework.

The probabilistic model of semantic similarity looks similar to the vector space model of semantic similarity, but its underpinnings are different. In a probabilistic semantic similarity model, a word’s context is represented as a proper probability distribution, obeying the laws of probability. Each feature on a word’s vector represents the probability $p(f \mid w)$, the probability of the feature given the word. This means that the original frequency matrix is normalized, so that each vector sums to 1, i.e. each feature value is divided by the sum of the vector’s feature values. If applied to the matrix in figure 2.1, this gives the matrix in figure 2.4.

<table>
<thead>
<tr>
<th></th>
<th>apple</th>
<th>banana</th>
<th>car</th>
<th>truck</th>
</tr>
</thead>
<tbody>
<tr>
<td>apple</td>
<td>1.000</td>
<td>.741</td>
<td>.164</td>
<td>.197</td>
</tr>
<tr>
<td>banana</td>
<td>.741</td>
<td>1.000</td>
<td>.103</td>
<td>.129</td>
</tr>
<tr>
<td>car</td>
<td>.164</td>
<td>.103</td>
<td>1.000</td>
<td>.996</td>
</tr>
<tr>
<td>truck</td>
<td>.197</td>
<td>.129</td>
<td>.996</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Figure 2.3: A word by word similarity matrix

<table>
<thead>
<tr>
<th></th>
<th>red_{adj}</th>
<th>yellow_{adj}</th>
<th>tasty_{adj}</th>
<th>fast_{adj}</th>
<th>eat_{obj}</th>
<th>drive_{obj}</th>
</tr>
</thead>
<tbody>
<tr>
<td>apple</td>
<td>.312</td>
<td>.037</td>
<td>.201</td>
<td>.000</td>
<td>.450</td>
<td>.000</td>
</tr>
<tr>
<td>banana</td>
<td>.002</td>
<td>.333</td>
<td>.191</td>
<td>.002</td>
<td>.469</td>
<td>.002</td>
</tr>
<tr>
<td>car</td>
<td>.177</td>
<td>.109</td>
<td>.000</td>
<td>.144</td>
<td>.001</td>
<td>.568</td>
</tr>
<tr>
<td>truck</td>
<td>.207</td>
<td>.136</td>
<td>.000</td>
<td>.115</td>
<td>.000</td>
<td>.542</td>
</tr>
</tbody>
</table>

Figure 2.4: A noun-by-features matrix normalized to probability $p(f \mid w)$

A number of similarity measures $S = \text{sim}(\vec{v}, \vec{w})$ are available to calculate the similarity between two probability vectors; the best known measure to calculate the similarity between probability distributions is the Kullback-Leibler (KL) divergence, which is defined as:
The KL divergence measures how well probability distribution $Q$ approximates probability distribution $P$; it tells us how much information we lose if we encode data with $Q$ when $P$ is the actual probability distribution.

There are, however, a number of problems with the KL divergence. First of all, it is undefined if there is a dimension $i$ with $Q(i) = 0$ and $P(i) \neq 0$. Secondly, the KL divergence is an asymmetric distribution, which means that $D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P)$.

There are other similarity measures that overcome these problems. The first one is the Jensen-Shannon (JS) divergence. The JS divergence is defined as:

$$D_{JS}(P \parallel Q) = \frac{1}{2}D_{KL}(P \parallel \frac{P + Q}{2}) + \frac{1}{2}D_{KL}(Q \parallel \frac{P + Q}{2})$$ (2.9)

Intuitively, the JS divergence tells us how much information is lost if the two probability distributions $P$ and $Q$ are replaced by the average of both distributions. The JS divergence does not have any problems with infinite values, and it is symmetric.

Another possibility is to approximate the KL divergence as close as possible by mixing it to a small degree with the other distribution. This is what the skew divergence does. The skew divergence is defined as:

$$D_{skew(\alpha)}(P, Q) = D_{KL}(P \parallel \alpha Q + (1 - \alpha)P)$$ (2.10)

The skew divergence constant $\alpha$ is a number between 0 and 1, usually set close to 1 to approximate the KL divergence as close as possible; a normal value of $\alpha = 0.99$. The measure remains asymmetric, but mixing in the other probability distribution to a small degree, effectively solves the infinity problem with zero values.

### 2.3 Weighting schemes

The methods described above can be applied to raw frequency counts. Often, though, an extra weighting step is applied in order to adapt the feature value according to its actual importance. In our formal model, this is the lexical association function or weighting function $A$. Many different weighting functions have been applied to the problem of semantic similarity. In the following paragraphs, we will have a look at the intuition behind them, and investigate the different possibilities.
2.3.1 Introduction: Zipf’s law

Zipf’s law states that the frequency of a word in any particular corpus is inversely proportional to its rank in a frequency list. As a result, word distributions are extremely skewed: the majority of words occur very infrequently, whereas the top few most frequent words take up the largest part of the corpus. This fact brings about a frequency bias: words with similar frequencies will be considered more similar than they actually are.

Intuitively, it is more significant for a word’s semantics to appear with an infrequent but highly specific, ‘meaningful’ feature than to appear with a very frequent, broad, ‘meaningless’ feature. As an example, compare denim skirt with nice skirt. It seems reasonable to attach more weight to the first feature denim than to the second feature nice. The former feature is highly specific and appears with a small subset of words (like denim pants, denim jeans, a denim jacket), whereas the latter is more broad and unspecific, and appears with a much larger set of words (a nice girl, a nice feeling, a nice zebra, ...). Moreover, a co-occurrence like denim skirt is much more informative than e.g. a skirt, although the latter one will have a much higher frequency. By applying a suitable weighting, we can neutralize the skewed frequencies arising from the Zipfian distribution.

Weighting functions can be divided into local and global weighting functions, according to the information they use in order to calculate the weighted value; local weighting functions only use a particular co-occurrence frequency count on its own to calculate the weighted value, whereas global weighting functions make use of global word and feature distribution statistics calculated over the corpus as a whole. In the following paragraphs, we will have a look at both types, and discuss their most important instantiations in the scope of semantic similarity.

2.3.2 Local weighting

A local weighting function is a function that is applied to a particular co-occurrence frequency without any knowledge about the corpus frequencies as a whole. In the simplest case, this amounts to applying the identity function to a particular co-occurrence frequency. Another simple local weighting is the application of a binary function, which assigns a value of one if the co-occurrence frequency is larger than zero (i.e. the combination occurs at least once in the corpus) and zero otherwise.

A local weighting function that is often used in the scope of semantic similarity is a logarithmic weighting function. A logarithmic weighting dampens large frequency values, so that frequent words are assigned less extreme values. The base of the
logarithm is often taken to be natural, though in practice the algorithm’s base does not influence the results.

\[ A_{\log}(f_{ij}) = 1 + \log(f_{ij}) \]  

for \( f_{ij} > 0 \). The function is represented graphically in figure 2.5.

![Figure 2.5: A logarithmic function is used to smooth extreme frequency values](image)

2.3.3 Global weighting

**Entropy**

In information theory, entropy is a measure to express the uncertainty (or surprise) that is associated with a random variable. Intuitively, if a particular word appears with only a few features (documents), it will be much more informative than a word that appears with lots of features (documents). Words that appear in many documents (i.e. words that are more uniformly distributed) will have a high entropy value. Words that appear in only a limited number of documents, on the other hand, will have a low entropy value.
Formally, entropy weighting is usually calculated according to the formula in 2.12.

\[ A_{\text{ent}}(i,j) = f_{ij}(1 + \sum_{k=1}^{n} \frac{p_{ik} \log(p_{ik})}{\log(n)}), p_{ik} = \frac{f_{ik}}{\sum_{l=1}^{n} f_{il}} \]  

(2.12)

with \( f_{ij} \) being the original co-occurrence frequency and \( n \) the total number of features (documents). This formula actually calculates an entropy ratio \( G(i) = 1 - \frac{H(d| i)}{H(d)} \), where \( H(d) \) is the entropy of the uniform distribution of the documents, and \( H(d | i) \) is the entropy of the conditional distribution given that the noun \( i \) appeared. The original co-occurrence frequency is then multiplied by this ratio. Intuitively, entropy weighting multiplies each noun’s feature vector by a constant vector that depends on the ‘informativeness’ of the noun.

Global entropy weighting can also be combined with local logarithmic weighting. The weighting of a particular value is then calculated according to the formula in 2.13.

\[ A_{\text{logent}}(i,j) = f_{ij}(1 + \sum_{k=1}^{n} \frac{p_{ik} \log(p_{ik})}{\log(n)}), p_{ik} = \frac{1 + \log(f_{ik})}{\sum_{l=1}^{n} 1 + \log(f_{il})} \]  

(2.13)

Entropy weighting (combined with logarithmic weighting) is the standard weighting model used in latent semantic analysis (LSA, cfr. infra). We will therefore include entropy as a weighting function in the document-based models. Note, however, that entropy weighting only makes sense in combination with a dimensionality reduction model such as LSA. Without any further processing, the entropy weighting will be undone by the normalisation of the feature vectors.

**Pointwise mutual information**

Another popular weighting function is called pointwise mutual information (PMI). PMI was first proposed by Church and Hanks (1990), and is based on the information-theoretic notion of mutual information. Mutual information measures the mutual dependence between two random variables \( X \) and \( Y \). It is defined as

\[ I(X,Y) = \sum_{x} \sum_{y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \]  

(2.14)

Pointwise mutual information – i.e. the mutual information for particular events – is defined as
\[ I(i, j) = \log \frac{p(i, j)}{p(i)p(j)} \] (2.15)

Intuitively, pmi tells us how much information a particular feature contains about a target word (and vice versa). pmi measures how often two events \( i \) and \( j \) occur, compared to the expected value if they were independent. The numerator gives the actual probability of the target word and feature occurring together, whereas the denominator contains the probability of the target word and feature occurring independently (multiplying the marginal probabilities). Thus, the ratio indicates how much more the target word and feature co-occur than we would expect by chance.

Although pmi is used a lot as weighting function, it is problematic for low frequency counts: the score will depend on the frequency of individual words. Thus, low-frequency co-occurrences will receive a higher score than high-frequency co-occurrences (all other things being equal). One solution to this problem is to use a particular cut-off (e.g. a co-occurrence frequency of at least 3). Another solution is the use of an extra weighting factor dependent on the frequency (Pantel and Lin, 2002).

We will evaluate pmi as a weighting function in the window-based and syntax-based models.