Chapter 2

Count-based Distributional Models

Keywords: count-based models, co-occurrence matrix, similarity measure, sparse vectors, bag-of-words, TF-IDF, (P)PMI, matrix factorization, SVD, LSA, LSI, PCA.

In this chapter, we will consider the models of distributional semantics which were originally introduced in the 1950s, but became widely used in 1970s when computers made it possible for researchers to process text data. In that time several programming languages such as C, Pascal, Fortran, etc. were developed. Also, first information systems were created, and that entailed the need in vector semantic space models.

2.1 First Vector Space Models: Matrix of Word Co-occurrence Counts

Vector or distributional models of meaning are generally based on a co-occurrence matrix, a way of representing how often words co-occur. This matrix can be constructed in various ways; let's s begin by looking at one such co-occurrence matrix.

2.1.1 Term-document Matrix

In a **term-document matrix**, each row represents a word in the vocabulary and each column represents a document from some collection of documents. Table 2.1 shows a small selection from a term-document matrix showing the occurrence of four words in four plays by Shakespeare. Each cell in this matrix represents the number of times a particular word (defined by the row) occurs in a particular document (defined by the column). Thus *fool* appeared 58 times in *Twelfth Night*.

	As You Like It	**Twelfth Night**	**Julius Caesar**	**Henry V**
battle	1	0	7	13
good	115	81	71	91
fool	37	58	2	5
wit	21	15	2	3

Table 2.1: The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

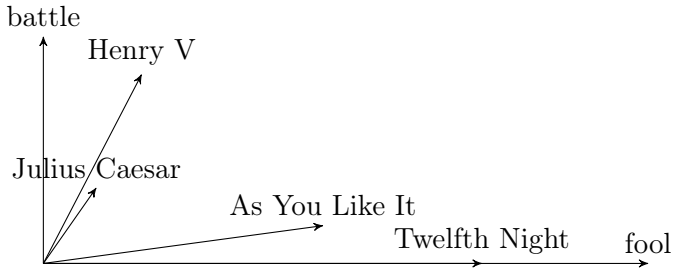

Figure 2.1: Vector space projection on 2-dimensional fool-battle space

The term-document matrix of Table 2.1 was first defined as part of the vector space model of information retrieval (Salton, 1971). In this model, a document is vector space model represented as a count vector, a column in Table 2.2.

So *As You Like It* is represented as the list $[1, 115, 37, 21]$ and *Julius Caesar* is represented as the list $[7, 71, 2, 2]$. In the example in Table 2.2, the vectors are of dimension dimension 4, just so they fit on the page; in real term-document matrices, the vectors representing each document would have dimensionality $|\mathcal{W}|$, the vocabulary size.

	As You Like It	**Twelfth Night**	**Julius Caesar**	**Henry V**
battle	1	0	7	13
good	115	81	71	91
fool	37	58	2	5
wit	21	15	2	3

Table 2.2: The term-document matrix for four words in four Shakespeare plays. The colored boxes show that each document is represented as a column vector of length four.

We can think of the vector for a document as identifying a point in $|\mathcal{W}|$-dimensional space; thus the documents in Table 2.2 are points in 4-dimensional space. Since 4-dimensional spaces are hard to draw in textbooks, Figure 2.1 shows a visualization in two dimensions; we've arbitrarily chosen the dimensions corresponding to the words *battle* and *fool*.

Term-document matrices were originally defined as a means of finding similar documents for the task of document information retrieval. Two documents that are similar will tend to have similar words, and if two documents have similar words their column vectors will tend to be similar. The vectors for the comedies *As You Like It* $[1, 115, 37, 21]$ and *Twelfth Night* $[0, 81, 58, 15]$ look a lot more like each other (more fools and wit than battles) than they do like *Julius Caesar* $[7, 71, 2, 2]$ or *Henry V* $[13, 91, 5, 3]$. We can see the intuition with the raw numbers; in the first dimension (*battle*) the comedies have low numbers and the others have high numbers, and we can see it visually in Pic. 2.3; we'll see very shortly how to quantify this intuition more formally.

A real term-document matrix, of course, wouldn't just have 4 rows and columns, let alone 2. More generally, the term-document matrix $\boldsymbol{W}$ has $|\mathcal{W}|$ rows (one for each word type in the vocabulary) and $|\mathcal{D}|$ columns (one for each document in the collection); as we'll see, vocabulary sizes are generally at least in the tens of thousands, and the number of documents can be enormous (think about all the pages on the web).

2.1.2 Term-term and Document-document Matrices

We have seen that documents can be represented as vectors in a vector space. However, semantic vectors can also be used to represent the meaning of words, by associating to each a vector.

The word vector is now a row vector rather than a column vector, and hence the row vector dimensions of the vector are different. The four dimensions of the vector for *fool*, $[37, 58, 2, 5]$, correspond to the four Shakespeare plays. The same four dimensions are used to form the vectors for the other 3 words: *wit*, $[21, 15, 2, 3]$; *battle*, $[1, 0, 7, 13]$; and *good* $[115, 81, 71, 91]$. Each entry in the vector thus represents the counts of the word's occurrence in the document corresponding to that dimension.

For documents, we saw that similar documents had similar vectors, because similar documents tend to have similar words. This same principle applies to words: similar words have similar vectors because they tend to occur in similar documents. The term-document matrix thus lets us represent the meaning of a word by the documents it tends to occur in.

However, it is most common to use a different kind of context for the dimensions of a word's vector representation. Rather than the term-document matrix we use the **term-term matrix**, more commonly called the **word-word matrix**, the **word-context matrix** or the **term-context matrix**, in which the columns are labeled by words rather than documents.

This matrix is thus of dimensionality $|\mathcal{W}| \times |\mathcal{W}|$ and each cell records the number of times the row (target) word and the column (context) word co-occur in some context in some training corpus. The context could be the document, in which case the cell represents the number of times the two words appear in the same document. It is most common, however, to use smaller contexts, generally a window around the word, for example of 4 words to the left and 4 words to the right, in which case the cell represents the number of times (in some training corpus) the column word occurs in such a ± 4 word window around the row word.

2.2 Measures of Word Co-occurrence

In order to build the word-context matrices, different possibilities are available and all of them do not require obligatory the use of $W_{i,j} = \#(w^{(i)}, c^{(j)})$ where $w^{(i)}$ denotes the ith word in our ordered or enumerated vocabulary $\mathcal{W}$, and the same can be said about the context $c^{(j)}$. Sometimes we associate words or contexts with their indices, so to obtain the simplicity in formulas we will write $W_{w,c} = \#(w, c)$.

In the following paragraph, the context is used in its general definition; it can be either a word, a sentence, a document, etc.

In this section, we will introduce several modifications and alternative ways of computing co-occurrence matrix.

2.2.1 Binary Matrix

In a binary matrix,

$$W_{w,c} = \operatorname{sgn} \#(w, c).$$

This means that in this case we are interested only in whether a word and a context appeared together. For instance, it can be used in hip-hop artists lexicon analysis.

2.2.2 TF-IDF

The term frequency—inverse document frequency or TF-IDF is a numerical statistics that tries to point out how important is a word in a given document. It is linked with the number of occurring and the number of documents contained in the corpora. The use of this function is very simple leading it to be often implemented in search engines.

The **term frequency (TF)** is proportional to the number of times a given query word w is found in the corpora. The simplest and most preferred choice is to use simple raw counts of a word w and a document d $\#(w, d)$:

$$TF_{w,d} = \#(w, d).$$

As it may appear, some words such as articles or prepositions can be used too often. In order to deal with this and not being flood by unnecessary information, an **inverse document frequency (IDF)** is also implemented. It is a factor increasing weights of elements and words seldom seen in a document. Thus for the elements of TF-IDF matrix we have the following formula:

$$W_{w,d} = TF_{w,d} \cdot IDF_w$$

where

$$IDF_w = \log \frac{|\mathcal{D}|}{|\{d \in \mathcal{D} : w \in d\}|}.$$

To simplify further formulas, we will denote $\mathcal{D}_{|w} := \{d \in \mathcal{D} : w \in d\}$. So,

$$IDF_w = \log \frac{|\mathcal{D}|}{|\mathcal{D}_{|w}|}.$$

The inverse document frequency was originally conceived by Karen Spärck Jones in 1972. See HIS PUBLICATION.

There are a lot of variants of this formula. For instance, $TF_{w,d}$ may be any reasonable function of $\#(w, d)$, e. g. $TF_{w,d} = \log(1 + \#(w, d))$, etc. Hence the formula has to be adapted. Another possibility is modification of IDF-term. Sometimes you can see formulas with shifted numerator and denominator. In general, the smoothed IDF can be computed as follows:

$$IDF_w^{\alpha,\beta} = \log \frac{|\mathcal{D}| + \alpha}{|\mathcal{D}_{|w}| + \beta}.$$

Anyway, the simplest version of TF-IDF weights has a probabilistic sense. Consider a model in which the probability of finding a word w in an arbitrary document d is

$$P(w) = \frac{|\mathcal{D}|}{|\mathcal{D}_{|w}|}.$$

Consider a query q which contains words $w_1, \ldots, w_Q$. To estimate the probability of finding words $w_1, \ldots, w_Q$ from a query q we can use a heuristic formula:

$$P(q, d) = \prod_{k=1}^{Q} P(w_k, d) = \prod_{k=1}^{Q} P(w_k)^{\#(w_k,d)} = \prod_{k=1}^{Q} \left(\frac{|\mathcal{D}|}{|\mathcal{D}_{|w_k}|} \right)^{\#(w_k,d)}.$$

Now let's take a logarithm of it:

$$\log P(q,d) = \sum_{k=1}^{Q} \#(w_k,d) \log \frac{|\mathcal{D}|}{|\mathcal{D}_{|w_k}|}$$

and multiply the result by -1. We will obtain

$$-\log P(q,d) = \sum_{k=1}^{Q} \#(w_k,d) \log \frac{|\mathcal{D}_{|w_k}|}{|\mathcal{D}|} = \sum_{k=1}^{Q} TF_{w_k,d} \cdot IDF_{w_k}.$$

Under the sum, the first term is a document frequency and the second term is an inverse document frequency.

While the implementation is easy, this algorithm does have some drawbacks. Basically, there is no semantic match to a given query. All synonyms or related terms can not be reached without modifications. Moreover, certain words may give some noisy results such as "apple" or other common words used by brands.

2.2.3 BM25

Okapi BM25 is function based on bag of words. Its role is to rank documents according to contained terms and relations between them. The name of "Okapi" refers to the first place where this function was implemented. Mainly used by searching engines, this algorithm overtakes the classic TD-IDF function from the speed side and the accuracy side. In 2015, Lucene, an open-source search engine, moved to BM25 with some modifications to increase its performances.

The way this function works is strongly linked with TF-IDF, however some important modifications are presented. For a given query q in a document d, all words are associated to some scoring functions and merged together at the end. The formula is as follows:

$$\begin{aligned}\text{BM25}(q,d) &= \sum_{k=1}^{Q} IDF_{w_k} \frac{\#(w_k,d)(k_1+1)}{\#(w_k,d) + k_1(1-b+b\frac{|\mathcal{D}|}{avgdl})} = \\ &= \sum_{k=1}^{Q} IDF_{w_k} \frac{TF_{w_k,d}(k_1+1)}{TF_{w_k,d} + k_1(1-b+b\frac{|\mathcal{D}|}{avgdl})}\end{aligned}$$

where $avgdl$ is the average document length and k_1 and b are two optimization parameters, usually $k_1 = 2.0$ and $b = 0.75$.

Nowadays, some modifications have been made to BM25 such has BM25F which splits documents into fields (head, core, etc.) before searching inside or BM25+ which adds one parameter to help BM25 to work with long documents.

2.2.4 PMI. Collocation Extraction

An alternative weighting function to TF-IDF is called PPMI (positive pointwise mutual information). PPMI draws on the intuition that best way to weigh the association between two words is to ask how much more the two words co-occur in our corpus than we would have a priori expected them to appear by chance. **Pointwise mutual information (PMI)** (Fano, 1961) is one of the most important concepts in NLP. It is a measure of how often

two random variables x and y occur, compared with what we would expect if they were independent:

$$I(\mathrm{x},\mathrm{y}) = \log_2 \frac{P(\mathrm{x},\mathrm{y})}{P(\mathrm{x})P(\mathrm{y})} = \log_2 \frac{P(\mathrm{x} \mid \mathrm{y})P(\mathrm{y})}{P(\mathrm{x})P(\mathrm{y})} = \log_2 \frac{P(\mathrm{x} \mid \mathrm{y})}{P(\mathrm{x})}$$
$$= \log_2 \frac{P(\mathrm{y} \mid \mathrm{x})P(\mathrm{x})}{P(\mathrm{x})P(\mathrm{y})} = \log_2 \frac{P(\mathrm{y} \mid \mathrm{x})}{P(\mathrm{y})}.$$

If $\mathrm{x} \perp \mathrm{y}$ then $P(\mathrm{x} \mid \mathrm{y}) = P(\mathrm{x})$ and $P(\mathrm{y} \mid \mathrm{x}) = P(\mathrm{y})$, and thus $I(\mathrm{x},\mathrm{y}) = 0$.

The pointwise mutual information between a target word w and a context word c (Church and Hanks 1989, Church and Hanks 1990) is then defined as:

$$PMI_{w,c} = \log_2 \frac{P(w,c)}{P(w)P(c)}.$$

The numerator tells us how often we observed the two words together (assuming we compute probability by using the MLE). The denominator tells us how often we would expect the two words to co-occur assuming they each occurred independently; recall that the probability of two independent events both occurring is just the product of the probabilities of the two events. Thus, the ratio gives us an estimate of how much more the two words co-occur than we expect by chance. PMI is a useful tool whenever we need to find words that are strongly associated.

PMI values range from negative to positive infinity. But negative PMI values (which imply things are co-occurring less often than we would expect by chance) tend to be unreliable unless our corpora are enormous. To distinguish whether two words whose individual probability is each 10^{-6} occur together more often than chance, we would need to be certain that the probability of the two occurring together is significantly different than 10^{-12}, and this kind of granularity would require an enormous corpus. Furthermore it's not clear whether it's even possible to evaluate such scores of "unrelatedness" with human judgments. For this reason it is more common to use **positive PMI (PPMI)** which replaces all negative PMI values with zero (Church and Hanks 1989, Dagan et al. 1993, Niwa and Nitta 1994):

$$PPMI_{w,c} = \max(PMI_{w,c}, 0) = PMI^{+}_{w,c}.$$

More formally, let's assume we have a co-occurrence matrix $\boldsymbol{W}$ with $|\mathcal{W}|$ rows (words) and $|\mathcal{C}|$ columns (contexts), where $W_{w,c} = \#(w,c)$. This can be turned into a PPMI matrix where

$$PPMI_{w,c} = \max(\log_2 \frac{p_{w,c}}{p_{w,:} \cdot p_{:,c}}, 0)$$

where

$$p_{w,c} = \frac{\#(w,c)}{\sum_{w' \in \mathcal{W}} \sum_{c' \in \mathcal{C}} \#(w',c')}, \; p_{w,:} = \sum_{c' \in \mathcal{C}} p_{w,c'}, \; p_{:,c} = \sum_{w' \in \mathcal{W}} p_{w',c}.$$

In computational linguistics, PMI has been used for finding collocations and associations between words. The following Table 2.3 shows counts of pairs of words getting the most and the least PMI scores in the first 50 millions of words in Wikipedia (dump of October 2015) filtering by 1,000 or more co-occurrences. The frequency of each count can be obtained by dividing its value by 50,000,952. (Note: natural log is used to calculate the PMI values in this example, instead of log base 2)

w	c	$\#(w)$	$\#(c)$	$\#(w,c)$	PMI
puerto	rico	1938	1311	1159	10.0349081703
hong	kong	2438	2694	2205	9.72831972408
los	angeles	3501	2808	2791	9.56067615065
carbon	dioxide	4265	1353	1032	9.09852946116
prize	laureate	5131	1676	1210	8.85870710982
san	francisco	5237	2477	1779	8.83305176711
nobel	prize	4098	5131	2498	8.68948811416
ice	hockey	5607	3002	1933	8.6555759741
star	trek	8264	1594	1489	8.63974676575
car	driver	5578	2749	1384	8.41470768304
it	the	283891	3293296	3347	-1.72037278119
are	of	234458	1761436	1019	-2.09254205335
this	the	199882	3293296	1211	-2.38612756961
is	of	565679	1761436	1562	-2.54614706831
and	of	1375396	1761436	2949	-2.79911817902
a	and	984442	1375396	1457	-2.92239510038
in	and	1187652	1375396	1537	-3.05660070757
to	and	1025659	1375396	1286	-3.08825363041
to	in	1025659	1187652	1066	-3.12911348956
of	and	1761436	1375396	1190	-3.70663100173

Table 2.3: The results of applying PMI to search collocations

Good collocation pairs have high PMI because the probability of co-occurrence is only slightly lower than the probabilities of occurrence of each word. Conversely, a pair of words whose probabilities of occurrence are considerably higher than their probability of co-occurrence gets a small PMI score.

PMI has the problem of being biased toward infrequent events; very rare words tend to have very high PMI values. One way to reduce this bias toward low frequency events is to slightly change the computation for $P(c)$, using a different function $P_\alpha(c)$ that raises contexts to the power of α:

$$PMI^\alpha_{w,c} = \log_2 \frac{P(w,c)}{P(w)P_\alpha(c)},\ P_\alpha(c) = \frac{\#(c)^\alpha}{\sum_{c' \in \mathcal{C}} \#(c')^\alpha}.$$

Levy et al. (2015) found that a setting of $\alpha = 0.75$ improved performance of embeddings on a wide range of tasks (drawing on a similar weighting used for skip-grams described below in Section 4). This works because raising the probability to $\alpha = 0.75$ increases the probability assigned to rare contexts, and hence lowers their PMI ($P_\alpha(c) > P(c)$ when c is rare).

Another possible solution is Laplace smoothing: Before computing PMI, a small constant k (values of $0.1 - 3$ are common) is added to each of the counts, shrinking (discounting) all the non-zero values. The larger the k, the more the non-zero counts are discounted.

2.3 Similarity Measures

2.3.1 Cosine Similarity

To define similarity between two target words u and v, we need a measure for taking two such vectors and giving a measure of vector similarity. By far the most common similarity metric is the **cosine** of the angle between the vectors.

$$\cos(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{\|\boldsymbol{u}\| \cdot \|\boldsymbol{v}\|} = \frac{\sum_{i=1}^{N} u_i v_i}{\sqrt{\sum_{i=1}^{N} u_i^2} \sqrt{\sum_{i=1}^{N} v_i^2}}.$$

Sometimes, to satisfy the first axiom of distance, we can use an equivalent formula:

$$\text{cosine}(\boldsymbol{u}, \boldsymbol{v}) = 1 - \cos(\boldsymbol{u}, \boldsymbol{v}).$$

2.3.2 Jaccard Index

The **Jaccard coefficient** measures similarity between finite sample sets $\mathbb{A}$ and $\mathbb{B}$, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(\mathbb{A}, \mathbb{B}) = \frac{|\mathbb{A} \cap \mathbb{B}|}{|\mathbb{A} \cup \mathbb{B}|}.$$

If $\mathbb{A}$ and $\mathbb{B}$ are both empty, we define $J(\mathbb{A}, \mathbb{B}) = 1$.

The Jaccard distance, which measures dissimilarity between sample sets, is complementary to the Jaccard coefficient and is obtained by subtracting the Jaccard coefficient from 1:

$$d_J(\mathbb{A}, \mathbb{B}) = 1 - J(\mathbb{A}, \mathbb{B}).$$

Given two objects, A and B, each with n binary attributes, the Jaccard coefficient is a useful measure of the overlap that A and B share with their attributes. Each attribute of A and B can either be 0 or 1. The total number of each combination of attributes for both A and B are specified as follows:

- M_{11} represents the total number of attributes where A and B both have a value of 1.
- M_{01} represents the total number of attributes where the attribute of A is 0 and the attribute of B is 1.
- M_{10} represents the total number of attributes where the attribute of A is 1 and the attribute of B is 0.
- M_{00} represents the total number of attributes where A and B both have a value of 0.

Then the Jaccard coefficient can be calculated as:

$$J(A, B) = \frac{M_{11}}{M_{11} + M_{01} + M_{10}}$$

and follows the Jaccard distance d_J :

$$d_J(A,B) = \frac{M_{10} + M_{01}}{M_{11} + M_{01} + M_{10}} = 1 - J(A,B).$$

Grefenstette in 19?? generalized these formulas on a case of non-negative real valued vectors $\boldsymbol{u}$ and $\boldsymbol{v}$.

$$\text{Jaccard}(\boldsymbol{u},\boldsymbol{v}) = \frac{\sum_{i=1}^{N} \min(u_i, v_i)}{\sum_{i=1}^{N} \max(u_i, v_i)}.$$

2.3.3 Dice Coefficient

The **Dice coefficient** is similar to the Jaccard index except that the denominator is the total weight of non zero entries within the two vectors compared.

$$\text{Dice}(\boldsymbol{u},\boldsymbol{v}) = \frac{2\sum_{i=1}^{N} \min(u_i, v_i)}{\sum_{i=1}^{N} u_i + v_i}.$$

An important point with the Dice coefficient is the use of wrong values. Instead of dealing only with positive ones, wrong values penalize the result. This leads to a different behavior than some other metrics.

2.3.4 Jensen—Shannon Divergence

TODO

2.4 Matrix Factorization. SVD, PCA

2.4.1 PCA

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components. If there are n observations with p variables, then the number of distinct principal components is $\min(n-1, p)$. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors (each being a linear combination of the variables and containing n observations) are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables.

PCA was invented in 1901 by Karl Pearson; it was later independently developed and named by Harold Hotelling in the 1930s.

PCA is the simplest of the true eigenvector-based multivariate analyses. Often, its operation can be thought of as revealing the internal structure of the data in a way that best explains the variance in the data. If a multivariate dataset is visualised as a set of

coordinates in a high-dimensional data space (1 axis per variable), PCA can supply the user with a lower-dimensional picture, a projection of this object when viewed from its most informative viewpoint[citation needed]. This is done by using only the first few principal components so that the dimensionality of the transformed data is reduced.

PCA is also related to canonical correlation analysis (CCA). CCA defines coordinate systems that optimally describe the cross-covariance between two datasets while PCA defines a new orthogonal coordinate system that optimally describes variance in a single dataset.

2.4.2 SVD

Matrix decomposition, also known as matrix factorization, involves describing a given matrix using its constituent elements.

Perhaps the most known and widely used matrix decomposition method is the **singular-value decomposition (SVD)**. All matrices have an SVD, which makes it more stable than other methods, such as the eigendecomposition. As such, it is often used in a wide array of applications including compressing, denoising, and data reduction.

SVD Theorem. Suppose $\boldsymbol{M}$ is a $m \times n$ matrix whose entries come from the field $\mathbb{K}$, which is either the field of real numbers $\mathbb{R}$ or the field of complex numbers $\mathbb{C}$. Then there exists a factorization, called a "singular value decomposition" of $\boldsymbol{M}$, of the form

$$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$$

where $\boldsymbol{U}$ is an $m \times m$ unitary matrix over $\mathbb{K}$ (if $\mathbb{K} = \mathbb{R}$, unitary matrices are orthogonal matrices), $\boldsymbol{\Sigma}$ is a diagonal $m \times n$ matrix with non-negative real numbers on the diagonal, $\boldsymbol{V}$ is an $n \times n$ unitary matrix over $\mathbb{K}$, and $\boldsymbol{V}^*$ is the conjugate transpose of $\boldsymbol{V}$.

The diagonal entries σ_i of $\boldsymbol{\Sigma}$ are known as the singular values of $\boldsymbol{M}$. A common convention is to list the singular values in descending order. In this case, the diagonal matrix, $\boldsymbol{\Sigma}$, is uniquely determined by $\boldsymbol{M}$ (though not the matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ if $\boldsymbol{M}$ is not square).

Some practical applications need to solve the problem of approximating a matrix $\boldsymbol{M}$ with another matrix $\tilde{\boldsymbol{M}}$, said truncated, which has a specific rank r. In the case that the approximation is based on minimizing the Frobenius norm of the difference between $\boldsymbol{M}$ and $\tilde{\boldsymbol{M}}$ under the constraint that rank$\tilde{\boldsymbol{M}} = r$ it turns out that the solution is given by the SVD of $\boldsymbol{M}$, namely

$$\tilde{\boldsymbol{M}} = \boldsymbol{U}\tilde{\boldsymbol{\Sigma}}\boldsymbol{V}^*$$

where $\tilde{\boldsymbol{\Sigma}}$ is the same matrix as $\boldsymbol{\Sigma}$ except that it contains only the r largest singular values (the other singular values are replaced by zero). This is known as the Eckart–Young theorem, as it was proved by those two authors in 1936 (although it was later found to have been known to earlier authors; see Stewart 1993).

For the case of simplicity we will focus on the SVD for real-valued matrices and ignore the case for complex numbers. Thus for every rectaingular matrix $\boldsymbol{M}$ there exist orthogonal matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ and a diagonal matrix $\boldsymbol{\Sigma}$ such that

$$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top.$$

2.5 Count-based Distributional Models Based on Matrix Factorization: LSA, LSI

2.5.1 LSA

Latent semantic analysis (LSA) is a theory and method for extracting and representing the contextual-usage meaning of words by statistical computations applied to a large corpus of text (Landauer and Dumais, 1997). The underlying idea is that the aggregate of all the word contexts in which a given word does and does not appear provides a set of mutual constraints that largely determines the similarity of meaning of words and sets of words to each other. The adequacy of LSA's reflection of human knowledge has been established in a variety of ways. Its performance has been assessed more or less rigorously in several ways. LSA was assessed as

1. a predictor of query-document topic similarity judgments.
2. a simulation of agreed upon word-word relations and of human vocabulary test synonym judgments.
3. a simulation of human choices on subject-matter multiple choice tests.
4. a predictor of text coherence and resulting comprehension.
5. a simulation of word-word and passage-word relations found in lexical priming experiments.
6. a predictor of subjective ratings of text properties, i.e. grades assigned to essays.
7. a predictor of appropriate matches of instructional text to learners.
8. LSA has been used with good results to mimic synonym, antonym, singular-plural and compound-component word relations, aspects of some classical word sorting studies, to simulate aspects of imputed human representation of single digits, and, in pilot studies, to replicate semantic categorical clusterings of words found in certain neuropsychological deficits.

2.5.2 LSI

J. R. Anderson (1990) has called attention to the analogy between **information retrieval** and human semantic memory processes. One way of expressing their commonality is to think of a searcher as having in mind a certain meaning, which he or she expresses in words, and the system as trying to find a text with the same meaning. Success, then, depends on the system representing query and text meaning in a manner that correctly reflects their similarity for the human. **Latent semantic indexing** (**LSI**; LSA's alias in this application) does this better than systems that depend on literal matches between terms in queries and documents. Its superiority can often be traced to its ability to correctly match queries to (and only to) documents of similar topical meaning when query and document use different words. In the text-processing problem to which it was first applied, automatic matching of information requests to document abstracts, SVD provides a significant improvement over prior methods. In this application, the text of the document database is first represented as a matrix of terms by documents (documents are usually represented by a surrogate such as a title, abstract and/or keyword list) and subjected to SVD, and each word and document

is represented as a reduced dimensionality vector, usually with 50-400 dimensions. A query is represented as a "pseudo-document" a weighted average of the vectors of the words it contains. (A document vector in the SVD solution is also a weighted average of the vectors of words it contains, and a word vector a weighted average of vectors of the documents in which it appears.)

The first tests of LSI were against standard collections of documents for which representative queries have been obtained and knowledgeable humans have more or less exhaustively examined the whole database and judged which abstracts are and are not relevant to the topic described in each query statement. In these standard collections LSI's performance ranged from just equivalent to the best prior methods up to about 30% better.

2.6 Clustering

In data analysis, **clustering**, or cluster analysis, is the process of creating sets of similar elements. This concept was made by Driver and Kroeber in anthropology and then used for classification of traits in psychology by Cattell in 1943.

Once clusters are made, the idea afterwards is to consider them either as references or as group of objects for data operations.

In the following lines, the uses of some clusters in natural language processing are going to be described.

2.6.1 Brown Clusters

The first model of cluster presented here is the **Brown clustering** model.

Consider a corpora as a single sequence of words $w_1, \ldots, w_T$. The Brown clustering algorithm groups elements of a vocabulary $\mathcal{W}$ into k classes $1, \ldots, k$ (0 is used for denoting the start of the sequence). It was initially proposed, in the natural language processing field, by Peter Brown et al. in (1). The main idea of Brown's model is that the words from one cluster often appear after words from other clusters.

Let C be the function which takes the word $w \in \mathcal{W}$ as input and returns its cluster number $0, \ldots, k$. Let E be a conditional probability distribution over words $\mathcal{W}$ given a cluster number of a current word, and Q be a probability distribution over cluster numbers given a cluster number of a previous word.

The algorithm tries to maximize the joint probability

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} E(w_t \mid C(w_t)) Q(C(w_t) \mid C(w_{t-1}))$$

(we will talk about generative models in the Chapter 3). The distributions E and Q and a function C are learned during the training procedure.

Friday Monday Thursday Wednesday Tuesday Saturday Sunday weekends Sundays Saturdays

June March July April January December October November September August

people guys folks fellows CEOs chaps doubters commies unfortunates blokes

down backwards ashore sideways southward northward overboard aloft downwards adrift

water gas coal liquid acid sand carbon steam shale iron

great big vast sudden mere sheer gigantic lifelong scant colossal

man woman boy girl lawyer doctor guy farmer teacher citizen

American Indian European Japanese German African Catholic Israeli Italian Arab

pressure temperature permeability density porosity stress velocity viscosity gravity tension

mother wife father son husband brother daughter sister boss uncle

machine device controller processor CPU printer spindle subsystem compiler plotter

John George James Bob Robert Paul William Jim David Mike

anyone someone anybody somebody

feet miles pounds degrees inches barrels tons acres meters bytes

director chief professor commissioner commander treasurer founder superintendent dean custodian

Figure 2.2: Brown clusters

The original algorithm has a number of modifications. For instance, if you run it again for each separate cluster, you will get hierarchical Brown clusters. The final tree, or cluster, is composed of a reduce number of roots and a large number of leafs (where all elements of the corpora can be found). Thanks to this architecture, cluster merging is rather simple.

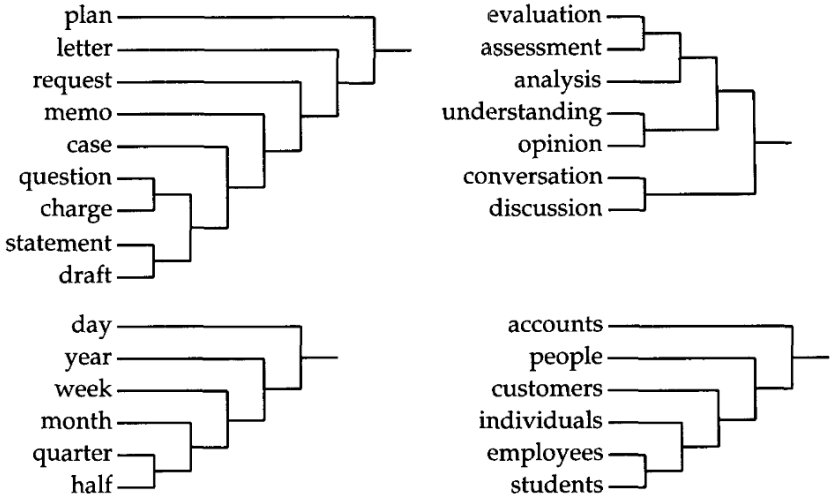

Figure 2.3: Hierarchical Brown clusters

One of the main drawbacks of this method is the limitation of range (original implementation requires $O(|\mathcal{W}|^5)$ operations, more efficient version—$O(|\mathcal{W}|^3)$, but it's still hard for real-world datasets). This can be seen if word-models are too simple, as it may be the case with bi-gram. For instance, it is mostly the case that only most common words from clusters

are going to be predicted. This limitation can be overtaken with more complex models but solutions stay in the field of greedy heuristic results.

2.6.2 Self-organizing Semantic Maps

Self-organizing maps (SOMs) (Ritter and Kohonen 1989, 1990) are a kind of neural network (see Chapter 4) that aims to reduce the dimension of a given input $\boldsymbol{x}$ through a mapping. This method is usually used as an unsupervised tool and works well for pattern recognition, statistical analysis and statistical similarities.

In natural language processing, the SOM can be used on contextual information to find similarities between words (the context in which they appear) and can organize them into grammatical and semantic categories represented as a two-dimensional array. These similarities of categories are calculated according to (normally Euclidean) distance of relationships within the array.

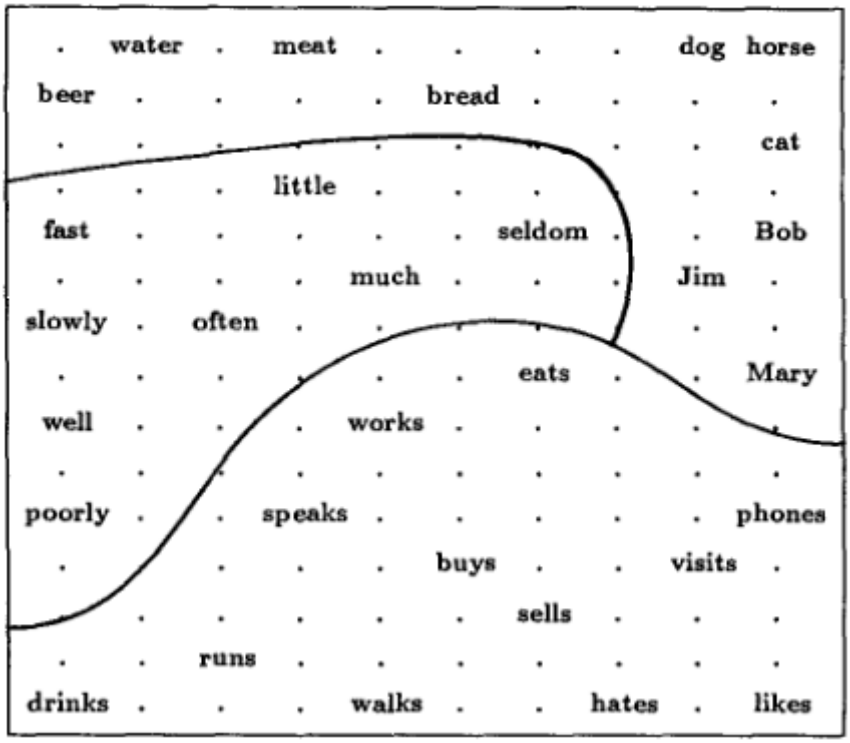

Figure 2.4: «Semantic map» obtained on a network of 10×15 cells after 2000 presentations of word-context-pairs derived from 10,000 random sentences of kind «Mary likes meat», «Jim speaks well», «Mary likes Jim», «Jim eats often»... Nouns, verbs and adverbs are segregated into different domains. Within each domain a further grouping according to aspects of meaning is discernible

In (4; 3), SOMs are reorganized according to word similarities and the SOM algorithm is based on competitive learning. The best matching neurons to inputs are selected and updated according to the adaptation rule.

Word category maps can be used in a large variety of tasks such information retrieval and text data mining as it has be shown that SOM can provide wield overviews of large corpora. Also, the competitive learning approach can be seen as a good basis for natural language interpretation.

2.6.3 Hyperspace Analogue to Language

The **hyperspace analogue to language (HAL)** also known as **semantic memory** was introduced by (8; 9). It is based on the idea that similar words may share similar meanings and may appear rather closely to each other.

The process is as follows. The goal is creation of a matrix where unique words are stored and represented as rows and columns. Number of occurrences close to the other unique words are counted and filled in the matrix.

As the result, co-occurring words have similar rows as weights are given according to the distance between the occurrence and the context: the smaller the distance, the higher is the weight.

For the sentence "The cat eats all cookies and all the meat", the co-occurring matrix, with a window of 3, is as follows:

	the	cat	eats	all	cookies	and	meat
the	0	0	0	0	1	2	0
cat	3	0	0	0	0	0	0
eats	2	3	0	0	0	0	0
all	1	2	3	1	2	3	0
cookies	0	1	2	3	0	0	0
and	0	0	1	2	3	0	0
meat	3	0	0	0	0	1	0

To extract vectors from this matrix, row and column of each words are joined. For instance, the vector for *cat* and *meat* are given by:

$$\boldsymbol{v}_{cat} = [3,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0;\ 0,\ 0,\ 3,\ 2,\ 1,\ 0,\ 0],$$
$$\boldsymbol{v}_{meat} = [3,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0;\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0].$$

With vectors as shown, it is simple to compare then thanks to their cosine or other geometrical metrics.

2.6.4 Phrase Clustering

With current algorithms and resources, clustering have helped to solve or improve results in many natural language processing tasks. However, in some particular fields, things seem stuck due to not strong enough model. It may be the case in named entity recognition or some information comparison tasks.

Lin and Wu in (7) describe another approach. Instead of using word based clusters, their idea is to consider sentences. Indeed, information contained inside these ones are bigger than information contained within words. However, with current ways of clustering, sentences can not be used. Hence, they decide, instead of recreating a complete model to adapt the K-Means one to sentences. This algorithm is a semi-supervised learning algorithm which takes as features word-clusters and phrase-clusters.

One strength of the algorithm is its possibility to scale easily first and to give good results. The main assumption was as for words; to consider that similar contexts give similar meanings.

The context feature for a phrase is based on the frequency counts of words in a given window of phrases. Then, frequency counts are translated into pointwise mutual information (PMI):

$$PMI(phrase, feature) = \log\left(\frac{P(phrase, feature)}{P(phrase)P(feature)}\right).$$

The clustering process is based on the K-Means clustering which assigns elements to a cluster according to a distance function.

2.7 Spectral Word Embeddings (advanced)

2.8 Pros and Cons of Count-based Models

To summarize previous parts, count-based models have been used for decades now and shown multiple strength as in their use as in their implementation. Hypothesis needed for such algorithms are very convenient and let a lot of people worked and explore possibilities. Different approaches are possible within this formalism and results are about a lot of topic.

In the following sections, main advantages and drawbacks of these models are going to be described.

2.8.1 Advantages

The first and biggest strength of count-based models is their age; they have been used and explored for decades. Hence, results found are for most of them very well known and accurate. The support for these algorithms covers an impressive range going from linear algebra to information theory and statistics.

Another strength is about data manipulation. Even if for large amount of texts and words these models may suffer, as soon as sets and corpora are small enough, for a given task, accuracy, efficiency and speed are going to be very high. For a large part of semantic analysis performed in industry, the use of these techniques remains against neural network based approaches.

2.8.2 Drawbacks

If the strength of advantages push the use of such models, some drawbacks still remain. The first one is linked with the interpretation of queries; words inside one are most of the time seen as bag-of-word. In other words, the order of entities in a sentence does not matter: "A cat is chasing a dog" is equivalent to "A dog is chasing a cat", "Bayern vs. PSG" is equivalent to "PSG vs. Bayern". This miss comprehension of the grammar structure is very important and blocks the access, with those algorithms, to some area of natural language processing.

Linked with the idea of "no grammar structure" encoded, issues about ambiguous words can be raised. Indeed, some words share the same writing for different senses as it is the case with polysemous words.

The second main drawback is about "known-words". During training of algorithms, a certain quantity of words is being processed leading to vectors and matrices. However, it may appear that some words do not belong to the training data set and then are considered as unknown during computational process. This issue is well known and is called out-of-vocabulary words. In such case, one variant is to recompute all matrices, however, it will

require some time. This problem is seen with new documents too. For a given set of texts, adding some others once computation done can lead to new expressions and issues. As said in the previous part, these models are working very well for closed dimension set of data where the amount of information is not too big. Otherwise, some out of range problems can appear.

2.9 Seminar

- Explanation of SVD.
- Hands-on tutorial with count-based models.
- **Homework:** implementation of a count-based model for information retrieval task.

Bibliography

[1] Brown, P. F., Desouza, P. V., Mercer, R. L., Pietra, V. J. D., and Lai, J. C. (1992). Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479.

[2] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407.

[3] Honkela, T. (1997). Self-organizing maps of words for natural language processing applications. In *Proceedings of the International ICSC Symposium on Soft Computing*, pages 401–407. Citeseer.

[4] Honkela, T., Pulkki, V., and Kohonen, T. (1995). Contextual relations of words in grimm tales analyzed by self-organizing map. In *Proceedings of ICANN-95, international conference on artificial neural networks*, volume 2, pages 3–7. EC2 et Cie Paris.

[5] Jurafsky, D. and Martin, J. H. (2014). *Speech and language processing*, volume 3. Pearson London.

[6] Landauer, T. K., Foltz, P. W., and Laham, D. (1998). An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284.

[7] Lin, D. and Wu, X. (2009). Phrase clustering for discriminative learning. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1030–1038. Association for Computational Linguistics.

[8] Lund, K. (1995). Semantic and associative priming in high-dimensional semantic space. In *Proc. of the 17th Annual conferences of the Cognitive Science Society, 1995.*

[9] Lund, K. and Burgess, C. (1996). Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior research methods, instruments, & computers*, 28(2):203–208.

[10] Turney, P. D. and Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37:141–188.