

The vector space model

A web search engine allows users to pose queries and returns documents from the world wide web that are relevant. Google is a famous example. In this course we will spend some time exploring how linear algebra might be used to build such an engine.

The *vector space model* encodes both *search terms* and *documents* as vectors. An indexed collection of documents is represented as a *term-by-document matrix*. The ij th entry of this matrix records how many times the i th search term appeared in the j th document.

For example, consider these articles and search terms related to computer music. The indexed collection is represented by the term-by-document matrix A .

d1	Foot tapping: A brief introduction to beat induction
d2	Tracking musical beats in real-time
d3	A model for musical rhythm
d4	Pattern processing in music
d5	An online algorithm for real-time accompaniment
d6	Following an improvisation in real time
d7	Style and music

t1	beat
t2	rhythm
t3	music
t4	pattern
t5	real-time
t6	algorithm

$$A = \begin{bmatrix} & d1 & d2 & d3 & d4 & d5 & d6 & d7 \\ t1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ t2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ t3 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ t4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ t5 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ t6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Now suppose we are interested in retrieving documents related to “real-time music algorithms.” We first represent this query as the vector

$$q = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

because q is the linear combination of the terms “real-time,” “music,” and “algorithm.”

The documents in our index are represented by the columns of the matrix A , and we wish to find the columns that are most similar to q . One straightforward way to compare two vectors is to take their cosine. When the cosine is 1, the vectors are pointing in the same direction and when the cosine is 0, they are pointing in perpendicular directions. (Since all of the entries of A and our query vector q are positive, the cosine will be nonnegative.) Hence, to find documents that match our query, we compute

$$\cos \theta_j = \frac{q^t A_j}{\|q\| \|A_j\|}$$

for each document $j = 1, \dots, 7$ where A_j is the j th column of A .

In our example, we have the following cosines.

j	d1	d2	d3	d4	d5	d6	d7
θ_j	0	0.6667	0.4082	0.4082	0.8165	0.5774	0.5774

Typically, we choose a *threshold value* T such that if $|\cos \theta_j| \geq T$ then document j is declared relevant and returned to the user. If we choose $T = 0.5$ in our example, then the query “real-time music algorithms” would return documents 2, 5, 6 and 7.

There are many additional features that can be added to this basic model including:

- Indexing by pairs, triples or larger consecutive groups of words rather than single search terms.

- Weighting the frequency of search terms rather than just recording the number of times each term appears in a document.
- Low-rank approximations of A can enable *latent semantic analysis* in which relationships between documents can be inferred even if the documents do not literally share the same search terms.
- Once a set of documents has been declared relevant for a given query using a methodology like the one above, an algorithm such as PageRank (developed by Page and Brin, the founders of Google) uses knowledge about *links between documents* to order the most relevant and authoritative documents first.

Latent semantic analysis

Latent semantic analysis is a technique to process the term-by-document matrix in order to produce a vector space of *concepts* that are related to the terms and documents. The concept space typically has fewer dimensions than the spaces of terms or documents and can be used to improve the query results.

Latent semantic analysis is carried out by performing a singular value decomposition (SVD) on the term-by-document matrix A and then constructing a low-rank approximation of A from the SVD. One of the most important properties of the SVD is that for any k , it produces the rank- k approximation to A with minimal error (in Frobenius norm). This is a theorem of Eckart and Young from 1936.

If the term-by-document matrix A is $m \times n$ then the singular value decomposition of this matrix is

$$A = USV^t$$

where U is $m \times m$, V is $n \times n$ and S is $m \times n$ containing the singular values on the diagonal entries. Let r be the number of non-zero diagonal elements of S . Then, r is the rank of A and so the first r columns of U form a basis for the column space of A . Recall that this column space contains the *documents* from our index. Similarly, the first r rows of V^t form a basis for the row space of A , which is the space of *search terms*. The decomposition describes how to transform vectors from the term or document spaces into vectors that lie in a space of “concepts.”

Specifically, if we choose $k \leq r$ then we obtain a rank k approximation to A by setting all but the k largest singular values in S to zero. Equivalently, we form

$$A_k = U_k S_k V_k^t$$

where U_k and V_k are formed from the first k columns of U and V , respectively, and S_k is the $k \times k$ diagonal matrix containing the k largest values from S .

The result of lowering the rank is that some of the dimensions must be combined. For example, suppose A contained three search terms “car,” “truck,” and “flower.” After lowering the rank to two, there is a basis for the concept space with two elements and we can express these elements as linear combinations of search terms. In the example, we might have the concepts “1.3 (car) + 0.3 (truck)” and “flower.” We can think of the first basis element as linear combination of “car” and “truck” that represents a concept like “personal transportation vehicle.”

It is instructive to consider the extreme case of $k = 1$. In this case, U_k contains a single column and V_k^t contains a single row. For example, the term-by-document matrix from the first section has rank-1 approximation given by

$$\begin{array}{c}
 \begin{bmatrix} 0.35 \\ 0.17 \\ 0.76 \\ 0.17 \\ 0.48 \\ 0.11 \end{bmatrix} \\
 [2.35] \\
 \begin{bmatrix} 0.15 & 0.68 & 0.39 & 0.39 & 0.25 & 0.20 & 0.32 \end{bmatrix}
 \end{array}
 =
 \begin{bmatrix} 0.1234 & 0.5593 & 0.3208 & 0.3208 & 0.2056 & 0.1645 & 0.2632 \\
 0.0599 & 0.2717 & 0.1558 & 0.1558 & 0.0999 & 0.0799 & 0.1278 \\
 0.2679 & 1.2145 & 0.6965 & 0.6965 & 0.4465 & 0.3572 & 0.5715 \\
 0.0599 & 0.2717 & 0.1558 & 0.1558 & 0.0999 & 0.0799 & 0.1278 \\
 0.1692 & 0.7670 & 0.4399 & 0.4399 & 0.2820 & 0.2256 & 0.3610 \\
 0.0388 & 0.1758 & 0.1008 & 0.1008 & 0.0646 & 0.0517 & 0.0827 \end{bmatrix}
 .$$

The vector U_k represents a single concept as a linear combination of search terms, and every column in A_k is a scalar multiple of this concept. The i th entry in the row vector V_k^t describes how closely document i corresponds to the concept. In general, the i th row of U describes the i th search term from A as a linear combination of k concepts, and the j th column of V^t gives the j th document as a linear combination of k concepts.

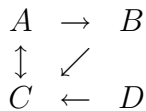
The original term-by-document matrix only lists the words that are actually in each document, but the low-rank approximation gives all words that are *related* to each document via concepts. Dimensions that are associated to terms with similar meanings are expected to merge in the SVD. These features can improve the accuracy of the search results.

The PageRank algorithm

The PageRank algorithm was developed by Larry Page and Sergey Brin at Stanford in 1998 just before they went on to found the search engine company Google. The idea is to *rank* a collection of N webpages using the link structure between them. Specifically, we model a random websurfer who:

- (1) (Usually) follows a link from the current page, with all links having an equal probability of being chosen.
- (2) (Occasionally) jumps to a new page, for example by using a bookmark. In the model, we assume that all pages are equally likely to be chosen.

For example, consider the link structure of the documents $\{A, B, C, D\}$ given by the directed graph below.



If we just consider (1), we have a random walk on a directed graph, and this can be modeled using a Markov chain. The state vectors are

$$x_n = \begin{bmatrix} x_n^{(1)} \\ \dots \\ x_n^{(N)} \end{bmatrix}$$

where $x_n^{(i)}$ is the probability of being on page i at time n . The transition matrix T is the adjacency matrix of the directed graph, normalized so that each column sums to 1. In our example, this is

$$T = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1/2 & 0 & 0 & 0 \\ 1/2 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Now we may let

$$x_n = Tx_{n-1} = T^n x_0,$$

and look for a steady-state vector x_* such that $Tx_* = x_*$. This is also known as the *dominant eigenvector* of T because x_* is an eigenvector with eigenvalue 1. By the theory of Markov chains, a steady-state exists and we can think of the components $x_*^{(i)}$ of the steady-state as the probability that our random web surfer will visit page i after performing the random walk for a long time. We use the steady-state vector to *rank* the pages $\{1, \dots, N\}$ according to these probabilities.

The only flaw in this plan is that page D is never visited at all because it is a *source* in the directed graph. This is where condition (2) comes in. We let q be the probability that our random web surfer jumps to a new page, where all N pages are equally likely to be chosen. This corresponds to the user typing a web address directly into the navigation bar, or selecting a previously stored bookmark. A typical value for q is 0.15.

Now our model becomes

$$x_n = (1 - q)Tx_{n-1} + q \begin{bmatrix} 1/N \\ 1/N \\ \dots \\ 1/N \end{bmatrix}$$

and we are still interested in determining the long-term behavior of the system. We can fix this up to be a Markov chain using the following steps:

- (1) Add a new component to the state vector that represents the probability of jumping to a new page:

$$x_n = \begin{bmatrix} \text{probability of visiting page 1} \\ \text{probability of visiting page 2} \\ \dots \\ \text{probability of visiting page } N \\ \text{probability of jumping} \end{bmatrix}.$$

- (2) Add an extra row and column to the transition matrix making sure

that all of the columns still sum to 1. In our example, this would be:

$$T = \begin{bmatrix} 0 & 0 & (1-q)1 & 0 & (1-q)1/N \\ (1-q)1/2 & 0 & 0 & 0 & (1-q)1/N \\ (1-q)1/2 & (1-q)1 & 0 & (1-q)1 & (1-q)1/N \\ 0 & 0 & 0 & 0 & (1-q)1/N \\ q & q & q & q & q \end{bmatrix}.$$

Since all of the columns still sum to 1 and all of the entries are nonnegative, this is still the transition matrix for a Markov chain and so a dominant eigenvector exists. This dominant eigenvector is an eigenvector associated to eigenvalue 1, and it represents the steady-state of our system. In the example above, the dominant eigenvector is

$$x_* = \begin{bmatrix} 0.3166 \\ 0.1664 \\ 0.3350 \\ 0.0319 \\ 0.1500 \end{bmatrix}$$

so we would rank these pages in the order (C, A, B, D) . This makes sense from the graph because C has many pages which reference it, so it is viewed by this algorithm as being an authoritative source and consequently it has high “PageRank.”

References

- [1] Michael W. Berry and Murray Browne, *Understanding Search Engines*, 2nd edition, SIAM (2005).
- [2] <http://en.wikipedia.org/wiki/PageRank>.
- [3] http://en.wikipedia.org/wiki/Latent_semantic_indexing.