

College of the Redwoods

<http://online.redwoods.cc.ca.us/instruct/darnold/laproj>

1/100

Matrices, Vector Spaces, and Information Retrieval

Steve Richards and Azuree Lovely



Purpose

Classical methods of information storage and retrieval inconsistent and lack the capability to handle the volume of information with the advent of digital libraries and the internet. The goal of this paper is to show how linear algebra, in particular the vector space model could be used to retrieve information more efficiently.



The need for Automated IR

In the past documents were indexed by authors titles, abstracts, key words, and subject classifications. To retrieve any one of these documents involves searching through a card catalogue manually, which incorporates the opinions of the user. Then if an abstract or key word list were not Provided, a professional indexer or cataloger could have written one equating to more uncertainties. But today,

- There are 60,000 new books printed annually in the United States.
- The Library of Congress maintains a collection of more than 17 million books and receives 7000 new ones daily.
- There are currently 300 million web pages on the internet, with the average search engine acquiring pointers to about 10 million daily.

Automated IR can handle much larger data bases without prejudice.





Complications with IR

- Language disparities of programmers and users
- Complexities of language itself such as polysemy and synonymy
- Accuracy and Inclusivity
- Term or phrase weighting



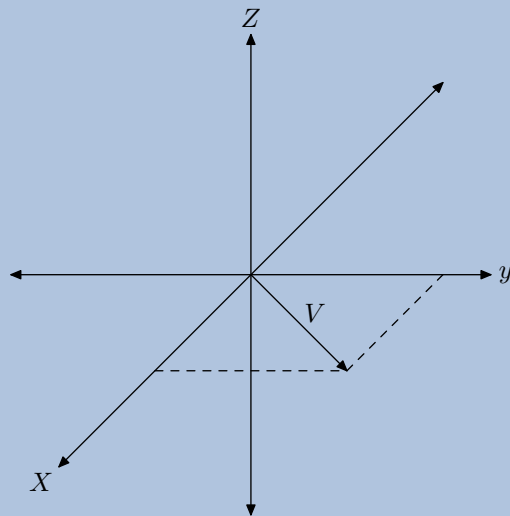
The Vector Space Model

Lets represent each document as a vector representing the relative frequency a term is used in that document. So the document “The Chevy Automobile: a Mechanical Marvel” will be indexed by the terms “Chevy”, “Auto”, and “Mechanic(s)”. The terms are identified by their roots, and any derivation of that root will be returned. The vector would be:

$$V = [1 \ 1 \ 0 \ 0 \ 1]^T$$



Graphically,



Query-vector comparison



An Example

Terms:

T_1 =auto(mobile,motive)

T_2 =Chevy

T_3 =Ford

T_4 =motor(s)

T_5 =mechanic(s,al)

Documents:

D_1 =The Chevy Automobile: A Mechanical Overview

D_2 =Automobiles Inside and Out

D_3 =The Ford Auto that rivaled Chevy's Chevelle

D_4 =A Mechanical Comparison of the motors of Chevy and Ford.

D_5 =A Mechanical Look at the motors in Chevy and Ford Automobiles





Now we describe our database by compiling the document vectors into the columns of a term by document matrix A , in which the rows are the term vectors.

$$A = \begin{pmatrix} T_1D_1 & T_1D_2 & T_1D_3 & T_1D_4 & T_1D_5 \\ T_2D_1 & T_2D_2 & T_2D_3 & T_2D_4 & T_2D_5 \\ T_3D_1 & T_3D_2 & T_3D_3 & T_3D_4 & T_3D_5 \\ T_4D_1 & T_4D_2 & T_4D_3 & T_4D_4 & T_4D_5 \\ T_5D_1 & T_5D_2 & T_5D_3 & T_5D_4 & T_5D_5 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

In order to weight each term in relevance to each document and also for query comparison, we normalize the matrix,

$$A = \begin{pmatrix} .7071 & 1 & .5774 & 0 & .4772 \\ 0 & 0 & .5774 & .5 & .4772 \\ 0 & 0 & .5774 & .5 & .4772 \\ 0 & 0 & 0 & .5 & .4772 \\ .7071 & 0 & 0 & .5 & .4772 \end{pmatrix}$$



Query comparison

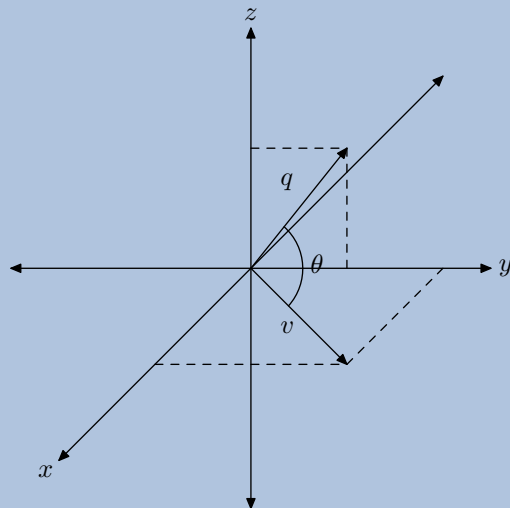
A query by a user will be represented as a vector in the same space. A user may query the database for Chevy motors, in which case the query vector would be $q = [0 \ 1 \ 0 \ 1 \ 0]^T$. The vectors in the database closest to that vector will be returned as relevant. This relevance is determined by the cosine of the angle between them:

$$\cos \theta = a_j^T q / \|a_j\| \|q\|.$$

Where $\|a_j^T\|$ is the Euclidian norm equal to $\sqrt{a^T a}$.



Graphically this comparison would look like,



Query-vector comparison



A threshold must be set for the minimum acceptable value for $\cos \theta$ of those documents returned to the user. The cosine of the angles between the document vectors in the database and the query vector are 0, 0, .4083, .7071, and .6324. This query would return the fourth and fifth documents, but the second may be the best resource and is not returned. The rest of this paper will be devoted to trying to resolve this problem.





Rank Reduction: Using QR Factorization

The next step is to make our system more efficient in handling mass amounts of information. The first step in doing so is to remove excess information, contained in the column space of A , that adds no new insight to the database. We can do this by identifying and ignoring dependencies. Reducing the rank of our term-document matrix can accomplish this, and one method for doing this is QR Factorization.

$$A=QR$$

$$R=t \times d \text{ upper triangular}$$

$$Q=t \times t \text{ orthogonal}$$

The relationship $A = QR$ says that the columns of A are linear combinations of the columns of Q . Therefore the columns of Q form a basis for the column space of A .



Returning to our example The factors would be:

$$Q = \left(\begin{array}{cccc|c} .7071 & .7071 & 0 & 0 & 0 \\ 0 & 0 & .7071 & 0 & 0 \\ 0 & 0 & .7071 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ .7071 & -.7071 & 0 & 0 & 0 \end{array} \right)$$

$$R = \left(\begin{array}{ccccc} 1 & .7071 & .4083 & .3536 & .6324 \\ 0 & .7071 & .4083 & -.3536 & 0 \\ 0 & 0 & .8166 & .7071 & .6324 \\ 0 & 0 & 0 & .5 & .4472 \\ \hline 0 & 0 & 0 & 0 & 0 \end{array} \right)$$

The zero row in R specifies that the last column in Q is a dependent one and can be ignored.





To reduce the rank of R , we block out matrix R . We get:

$$\left(\begin{array}{ccc|cc} 1 & .7071 & .4083 & .3536 & .6324 \\ 0 & .7071 & .4083 & -.3536 & 0 \\ 0 & 0 & .8166 & .7071 & .6324 \\ \hline 0 & 0 & 0 & .5 & .4472 \\ 0 & 0 & 0 & 0 & 0 \end{array} \right) = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} = \hat{R}$$

Because setting R_{22} equal to zero only produces a 30% change in matrix R , the new reduced rank matrix \hat{R} could be a good approximation to R .



By $A=QR$, the new matrix \hat{A} is:

$$\hat{A} = \begin{pmatrix} .7071 & 1 & .5774 & 0 & .4472 \\ 0 & 0 & .5774 & .5 & .4472 \\ 0 & 0 & .5774 & .5 & .4472 \\ .7071 & 0 & 0 & .5 & .4472 \end{pmatrix}$$

Calculating $\cos \theta$ between the query and the new matrix \hat{A} returns values of 0, 0, .4083, .4083, and .3953. Therefore the change in A was too large. Sometimes this may be the case, which is why we need to find a better means of obtaining a low rank approximation to matrix A .





Rank Reduction: Using Singular Value Decomposition

QR Factorization identifies dependencies in columns of matrix A , removing excess information from the system. However, dependencies in the row space must also be addressed. SVD is one method used for 1) removing those dependencies, 2) for producing a low rank approximation to A , and 3) for comparing terms to terms in the database.

$$A = U \Sigma V^T$$

$U = t \times t$ orthogonal

$\Sigma = t \times d$ diagonal

$V = d \times d$ orthogonal

Where U contains the column space of A , V contains the row space of A , and Σ contains the singular values of matrix A .



We can now reduce the rank of A to $A_k = U_k \Sigma_k V_k^T$ by setting all but the k largest singular values of A equal to zero. Returning to our previous example:

$$\Sigma = \left(\begin{array}{ccc|cc} 1.7873 & 0 & 0 & 0 & 0 \\ 0 & 1.0925 & 0 & 0 & 0 \\ 0 & 0 & .7276 & 0 & 0 \\ \hline 0 & 0 & 0 & .2874 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array} \right)$$





Thus using SVD produces only a 13% change in A. Comparing this to the 30% change in A produced by QR Factorization, it can be seen that SVD has the potential to produce better approximation to A. Doing so:

$$\hat{A} = \begin{pmatrix} .7293 & .9761 & .6013 & -.0070 & .4302 \\ -.0303 & .0326 & .5447 & .5096 & .4704 \\ -.0303 & .0326 & .5447 & .5096 & .4704 \\ .1250 & -.1346 & .1349 & .4603 & .3515 \\ .6558 & .0552 & -.0553 & .5163 & .4865 \end{pmatrix}$$

The cosines of the angles between the example query vector and this new approximation to A are .1098, -.0721, .4805, .6858, and .5811. Since the fourth and fifth documents are returned we have a successful reduced rank version of A.



Conclusion

QR Factorization removed dependencies in the column space of A , but could not, in this case, reduce the rank of A without losing information. SVD not only removed the dependencies in the column space, and also from the row space of A , but it also successfully reduced the rank of A . With these new tools, the vector space model can be used effectively in Information Retrieval.

