October 27, 2016

CSCI 2820
Application 2, part 2: Refining the Search Engine

Due Thursday, November 10

To complete this part of the assignment, you will need the search engine script you built in part 1 (or use the one from the solutions). Recall that the search engine returns the indices of the columns of the term-by-document matrix corresponding to documents that are similar to a given query. The measure of similarity is the cosine of the angle between the query vector q and the document vector a_j defined by:

$$\cos\theta_j = \frac{a_j^T q}{\| a_j \|_2 \| q \|_2}. \tag{1}$$

Recall also that the results for some of our test queries were unsatisfactory. We now find out how to improve those results and so to improve the quality of the search engine in general. In particular, we will use the QR decomposition to identify and remove redundant information in the matrix representation of the database. The rest of this document outlines the steps of that procedure.

Identifying redundancy in the database

Redundancy in the database is signalled by a lowered rank of its term-by-document matrix representation. Note that the 6×5 term-by-document matrix of the example in Figure 2 of part 1 is of rank four because column five is the sum of columns two and three. In this case, the first four columns of the matrix contain all of the semantic information in the database. Any document in the database can be reconstructed from those vectors. In linear algebra terms, those four columns constitute a basis for the column space of the matrix.

Even greater dependence can be expected in practice: a database of library materials can contain different editions of the same book and a database of Internet sites can contain several mirrors of the same web page. As in our example, dependencies can also involve more than simple copies of information: binary vectors representing the documents **applied linear algebra** and **computer graphics** sum to the binary vector representing **linear algebra applied to computer graphics** (where the preposition *to* is not considered to be a term), so any database containing all three documents

would have dependencies among its columns. In that case, a more compact (and "cleaner") representation of the semantic content of the database would be given by any basis for its column space.

In the next section, we show how to find a set of basis vectors for the column space of any term-by-document matrix.

Identifying a basis for the column space

The first step in identifying and removing redundant information in the database is to identify dependence between the columns or rows of the $t \times d$ term-by-document matrix. For a rank r_A matrix A, the r_A basis vectors of its column space serve in place of its d column vectors to represent its column space. One set of basis vectors is found by computing the QR decomposition of the term-by-document matrix using Gram-Schmidt orthogonalization

$$A = QR,$$

where R is a $d \times d$ upper triangular matrix and Q is a $t \times d$ matrix with orthonormal columns. This decomposition exists for any matrix A. The relation $A = QR$ shows that the columns of A are all linear combinations of the columns of Q. Thus, a subset of r_A of the columns of Q forms a basis for the column space of the rank r_A matrix A.

We now demonstrate how to identify the basis vectors of the example term-by-document matrix A from Figure 2 by using the QR decomposition. If $A = QR$, the factors are

$$Q = \left(\begin{array}{rrrrr} -0.5774 & 0 & -0.4082 & 0 & -0.7071 \\ -0.5774 & 0 & 0.8165 & 0 & 0.0000 \\ -0.5774 & 0 & -0.4082 & 0 & 0.7071 \\ 0 & 0 & 0 & -0.7071 & 0 \\ 0 & -1.0000 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.7071 & 0 \end{array}\right), \quad (2)$$

$$R = \left(\begin{array}{rrrrr} -1.0001 & 0 & -0.5774 & -0.7070 & -0.4082 \\ 0 & -1.0000 & 0 & -0.4082 & -0.7071 \\ 0 & 0 & 0.8165 & 0 & 0.5774 \\ 0 & 0 & 0 & -0.5774 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \end{array}\right). \quad (3)$$

It is not hard to show that Q is a rank four matrix where the fifth column is a linear combination of the first and third. In Equation (2), we have partitioned the matrix Q to separate the first four column vectors from the last, dependent column. In Equation (3), we have partitioned the matrix R to separate the 4×5 nonzero part from the 1×5 zero part. We now rewrite the decomposition $A = QR$ as

$$\begin{aligned} A &= \begin{pmatrix} Q_A & Q_A^{\perp} \end{pmatrix} \begin{pmatrix} R_A \\ 0 \end{pmatrix} \\ &= Q_A R_A + Q_A^{\perp} \cdot 0 = Q_A R_A, \end{aligned} \tag{4}$$

where Q_A is the 6×4 matrix holding the first four columns of Q, $Q_A^{\perp}$ is the 6×1 remaining submatrix of Q, and R_A covers the nonzero rows of R. This partitioning clearly reveals that the column of $Q_A^{\perp}$ does not contribute to the value of A and that the ranks of A, R, and R_A are equal. Thus, the four columns of Q_A constitute a basis for the column space of A.

Because the semantic content of a database is fully described by any basis for the column space of the associated term-by-document matrix, query matching can proceed with the factors QR in place of the matrix A. The cosines of the angles θ_j between a query vector q and the document vectors a_j are then given by

$$\cos\theta_j = \frac{a_j^T q}{\| a_j \|_2 \| q \|_2} = \frac{(Q_A r_j)^T q}{\| Q_A r_j \|_2 \| q \|_2} = \frac{r_j^T (Q_A^T q)}{\| r_j \|_2 \| q \|_2}, \tag{5}$$

for $j = 1, \ldots, d$. In this relation, we have used the fact that multiplying a vector by any matrix with orthonormal columns leaves the vector norm unchanged, i.e.,

$$\| Q_A r_j \|_2 = \sqrt{(Q_A r_j)^T Q_A r_j} = \sqrt{r_j^T Q_A^T Q_A r_j} = \sqrt{r_j^T I r_j} = \sqrt{r_j^T r_j} = \| r_j \|_2.$$

We now revisit the example term-by-document matrix from Figure 2 using the query vector $q^{(1)}$ (**baking bread**) and observe that there is no loss of information from using its factored form. As expected, the cosines computed via Equation (5) are the same as those computed using Equation (1): 0.8165, 0, 0, 0.5774, and 0.

Note that the rank reduction steps allow us to set portions of the matrix to zero and thus to ignore them in subsequent computations. Doing so lowers the cost of query matching and helps to recoup some of the expense of computing the decomposition.

The Low-Rank Approximation

Up to this point, we have used the QR factorization to explain how to remove obvious redundancies in the database. In addition, the QR factorization gives us a means of dealing with uncertainties in the database. Just as measurement errors can lead to uncertainty in experimental data, the very process of indexing the database can lead to uncertainty in the term-by-document matrix. A database and its matrix representation may be built up over a long period of time, by many people with different experiences and different opinions about how the database content should be categorized.

For instance, in the example of Figure 2, one could argue that the fifth document is relevant to *baking* since it is about **pastry recipes** which are simply instructions for **baking pastry**. Under that interpretation the (unnormalized) term-by-document matrix $\hat{A}$ would have the entry $\hat{A}_{15} = 1$. Because the best translation from data to matrix is subject to interpretation, a term-by-document matrix A might be better represented by a matrix sum $A + E$, where the *uncertainty* matrix E may have any number of values reflecting missing or incomplete information about documents or even different opinions on the relevancy of documents to certain subjects.

Now, if we accept the fact that our matrix A is only one representative of a whole family of relatively close matrices representing the database, it is reasonable to ask if it makes sense to attempt to determine its rank exactly. For instance, if we find the rank r_A and, using linear algebra, conclude that changing A by adding a *small* change E would result in a matrix $A + E$ of lesser rank k, then we may as well argue that our problem has a rank-k matrix representation and that the column space of A is not necessarily the best represention of the semantic content of the database. Next we show how lowering the rank may help to remove extraneous information or noise from the matrix representation of the database. (Remember also that the lower the value of k, the more computational savings we see.)

To proceed, we need a notion of the *size* of a matrix. In particular, we need to be able to say when a matrix is *small* in comparison to another matrix. If we generalize the Euclidean vector norm (the norm we have covered in class) to matrices the result is the so-called Frobenius matrix norm which is defined for the real $t \times d$ matrix X by

$$\|X\|_F = \sqrt{\sum_{i=1}^{t} \sum_{j=1}^{d} x_{ij}^2}. \tag{6}$$

The Frobenius norm can also be defined in terms of the matrix trace $\text{Trace}(X)$

which equals the sum of the diagonal elements of the matrix $X^T X$:

$$\|X\|_F = \sqrt{\text{Trace}(X^T X)} = \sqrt{\text{Trace}(X X^T)}. \tag{7}$$

Using the latter definition, we show that premultiplying the matrix X by a $d \times t$ matrix with orthogonal rows O leaves the Frobenius norm unchanged:

$$\|OX\|_F = \sqrt{\text{Trace}((OX)^T(OX))} = \sqrt{\text{Trace}(X^T O^T O X)} = \sqrt{\text{Trace}(X^T X)} = \|X\|_F.$$

Our aim is to find a reasonable low rank approximation to the matrix A. We focus on the upper triangular matrix R, using the fact that the ranks of R and A are equal. While the rank of A is not generally obvious, the rank of R is easy to determine as it is equal to the number of nonzero entries on its diagonal. The QR factorization aids us in manipulating the rank of R as it tends to separate the large and small parts of the matrix, pushing the larger entries toward the upper left corner of the matrix and the smaller ones toward the lower right. If this separation is successful, the matrix R can be partitioned to isolate the small part. For example, the factor R for our example problem can be partitioned as follows:

$$R = \left(\begin{array}{rrr|rr} -1.0001 & 0 & -0.5774 & -0.7070 & -0.4082 \\ 0 & -1.0000 & 0 & -0.4082 & -0.7071 \\ 0 & 0 & 0.8165 & 0 & 0.5774 \\ \hline 0 & 0 & 0 & -0.5774 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right) = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}.$$

Under this partitioning, the submatrix R_{22} is a relatively small part of the matrix R. Specifically, $\| R_{22} \|_F / \| R \|_F = 0.5774/2.2361 = 0.2582$.

We now create a new upper triangular matrix $\tilde{R}$ by setting the small matrix R_{22} equal to the zero matrix. The new matrix $\tilde{R}$ has rank three, as does the matrix $A + E = Q\tilde{R}$. The uncertainty matrix E is then given by the difference

$$\begin{aligned} E &= (A + E) - A \\ &= Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix} - Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \\ &= Q \begin{pmatrix} 0 & 0 \\ 0 & -R_{22} \end{pmatrix}. \end{aligned}$$

Note that $\| E \|_F = \left\| \begin{pmatrix} 0 & 0 \\ 0 & -R_{22} \end{pmatrix} \right\|_F = \| R_{22} \|_F$. Because $\| A \|_F = \| Q^T A \|_F = \| R \|_F$, $\| E \|_F / \| A \|_F = \| R_{22} \|_F / \| R \|_F = 0.2582$. In words, making a

26% relative change in the value of R makes the same sized change in A, and that change reduces the ranks of both matrices by one.

Studies have shown that uncertainties of roughly this order may be introduced simply by disagreement between indexers. Thus, we may deem it acceptable to use the rank-3 approximation $A + E$ in place of the original term-by-document matrix A for query matching. If we compute the cosines using Equation (5), we need never compute the matrix $A + E$ explicitly but rather can use, from its QR factors, the first three columns of Q and the triangular matrix $\tilde{R}$ which has three zero rows.

To verify that we have not caused undue loss of accuracy, we return to the example of Figure 2 using the matrix $A + E$ in place of the original term-by-document matrix A. The cosines computed for query $q^{(1)}$ (**baking bread**) are 0.8165, 0, 0, 0.7071, and 0, and the cosines computed for query q_2 (**baking**) are 0.5774, 0, 0, 0.5000, and 0. In both of these cases, the results are actually improved, meaning that our rank-3 approximation $A+E$ appears to be a better representation of our database than is the original term-by-document matrix A.

To push the rank reduction farther, we repartition the matrix R so that its third row and column are also included in R_{22}. In this case, $\| R_{22} \|_F / \| R \|_F = 0.5146$, and discarding R_{22} to create a rank-two approximation of the term-by-document matrix introduces a 52% relative change in that matrix. The cosines for $q^{(1)}$ are now 0.8165, 0, 0.8165, 0.7071, and 0.4082 and for q_2 are 0.5774, 0, 0.5774, 0.5000, and 0.2887. In both cases, some irrelevant documents are incorrectly identified, meaning that the 52% relative change in R and A is unacceptably large.

In general, it is not possible to explain why one variant of the term-by-document matrix works better than another for a given set of queries. We have seen, however, that it can be possible to improve the performance of the method by reducing the rank of the term-by-document matrix. Note that even the 26% change that we've chosen as acceptable in our example is quite large in the context of scientific or engineering applications where accuracies of three or more decimal places (0.1% error or better) are typically required.

The Assignment

1. Modify your search engine to use the QR factorization to implement rank reduction of the term-by-document matrix.

   You'll need to write your own QR routined (one of the Gram-Schmidts). Do not use the MATLAB qr command. It uses something called column pivoting which resembles the partial pivoting we looked at for $Ax = b$. The column pivoting will screw up your results.

   Further modify your search engine so that it returns the full list of cosines along with the list of indices of columns having cosines greater than the tolerance.

2. Using your revised search engine, test the following queries for ranks 1, 2, 3, and 4:

   $q^{(1)}$ Baking bread.

   $q^{(2)}$ Baking.

   $q^{(3)}$ What is the difference between pastry and bread?

   $q^{(4)}$ Show me all recipes for pastry.

   $q^{(5)}$ A query of your devising. (Specify.)

   Summarize your results in a clear and well-designed table.

   What matrix rank and cosine cutoff do you recommend for a search engine based on the vector space model using the QR decomposition for rank reduction? Your search engine will likely not operate perfectly, so you'll need to identify the best compromise between irrelevant documents returned and relevant documents missed.

   Your choices should be defended with about half a page of clear, consistent, and well-reasoned argument. Perfect spelling and reasonable grammar are expected. It is ok to repeat things you wrote in the first part of the assignment to support your argument.

How to choose the best rank for a given term-by-document matrix is an open research question, so your results are contributing to the understanding of it!

You may write the programs in this assignment with one partner. Partners may turn in a single paper with both names on it. Please put both names together at the top of the paper.

Here is a **recipe** for how to compute the QR factorization $A = QR$:

1. Implement the Gram-Schmidt process given as Theorem 11 in Section 6.4 of the text. The vectors $v_1, \ldots, v_p$ are the columns of Q.
2. Normalize the columns of Q.
3. Multiply A by Q^T to get R.

Check your answers: R should be upper triangular, Q^TQ should be the identity, QR should equal A.

Note that the Gram-Schmidt process is a doubly nested loop with a projection operation inside of it. If you take advantage of vector operations in MATLAB, it's only a few (maybe 5) lines of code.

To get a rank k approximation of A_k A, Set all but the first k rows of R equal to zero to get a rank k approximation to R called R_k. Then $A_k = QR_k$.