Information Retrieval Through Various Approximate Matrix Decompositions

Kathryn Linehan, klinehan@math.umd.edu
Advisor: Dr. Dianne O'Leary, oleary@cs.umd.edu

Abstract

In short, information retrieval is extracting certain information from databases. The purpose of this project is to explore querying a document database. We investigate and analyze the use of various approximate matrix decompositions to accomplish this task.

1 Background

The world is full of information, which would not be useful to us if we did not have a way of organizing and extracting it. In short, information retrieval is extracting certain information from databases. One example of information retrieval is the web search engine Google. The main issues associated with information retrieval are storage space, speed, and how "good" the results are, where "good" is a task specific measure. The objective of this project is to investigate querying a document database. A secondary, time permitting objective is to investigate forming a multidocument summary.

1.1 Querying a Document Database

Given a document database, we want to be able to return documents that are relevant to given query terms. There are existing solutions to this information retrieval problem, such as literal term matching and latent semantic indexing (LSI). Background information and examples presented in this section are taken from [5].

1.1.1 Problem Formulation

In real systems, such as Google, this problem is formulated in terms of matrices. An $m \times n$ term-document matrix, A, is created "where entry a_{ij}

represents the weight of term i in document j". Thus, each row in A represents a term and each column in A represents a document. Also, an $m \times 1$ query vector, q is created "where q_i represents the weight of term i in the query". Different weighting schemes for A and q are discussed in [5].

1.1.2 Literal Term Matching

In literal term matching, a relevance score is computed for each document as an inner product between q^T and the column of A that represents that document. The highest scoring documents are then returned.

The original term-document matrix is generally sparse; thus, unfortunately, literal term matching may not return relevant documents that contain synonyms of query terms, but not the actual query terms. We present an example of this below.

Example: Literal Term Matching

	Document				
Term	1	2	3	4	**Query**
Mark	15	0	0	0	1
Twain	15	0	20	0	1
Samuel	0	10	5	0	0
Clemens	0	20	10	0	0
Purple	0	0	0	20	0
Fairy	0	0	0	15	0
Score	30	0	20	0	

As seen in the above example, we have queried for the terms "Mark Twain". We notice that document 2 does not contain the terms "Mark Twain", but does contain the terms "Samuel Clemens" (who is the same person as Mark Twain) and is therefore relevant to the query. However, it has a relevance score of zero and thus is not returned as relevant. We would like to have a document querying system in which this problem is solved.

1.1.3 Latent Semantic Indexing

In latent semantic indexing (LSI), an approximation to the term-document matrix is used to compute document relevance scores. Using a matrix approximation can introduce nonzero entries, possibly revealing relationships between synonyms, and thus relevant documents that may not have the exact query terms may be returned.

A commonly used approximate matrix decomposition in LSI is a rank-k singular value decomposition (rank-k SVD). Below, we present an example of LSI using a rank-2 approximation to the term-document matrix presented in the literal term matching example.

Example: Latent Semantic Indexing

	Document				
Term	1	2	3	4	**Query**
Mark	3.7	3.5	5.5	0	1
Twain	11.0	10.3	16.1	0	1
Samuel	4.1	3.9	6.1	0	0
Clemens	8.3	7.8	12.2	0	0
Purple	0	0	0	20	0
Fairy	0	0	0	15	0
Score	14.7	13.8	21.6	0	

We see that in this example, by using LSI, document 2 now has a score of 13.8 and will therefore be returned as the third most relevant document, even though it did not contain the exact query terms "Mark Twain". This is an improvement over the relevance results of the literal term matching example.

In [5] a rank-k semidiscrete decomposition (SDD) is used in LSI. The rank-k SDD of an $m \times n$ matrix A is $A_k = X_k D_k Y_k^T$, where X_k is $m \times k$, D_k is $k \times k$, and Y_k is $n \times k$. Each entry of X_k and Y_k is one of $\{-1, 0, 1\}$ and D_k is diagonal with positive entries. The SDD is discussed in detail in [5].

In [5], an implementation of the SDD written by Tamara G. Kolda and Dianne P. O'Leary is used to determine that LSI using the SDD performs as well as LSI using the SVD, but does so using less storage space and query time.

An objective of this project is to test the performance of other approximate matrix decompositions when used in LSI.

1.1.4 Performance Measurement

To determine how well a document retrieval system performs, we use two measurements of performance: precision and recall. We define the following variables:

$Retrieved$ = number of documents retrieved
$Relevant$ = total number of relevant documents to the query
$RetRel$ = number of documents retrieved that are relevant.

Precision is defined as

$$P(Retrieved) = \frac{RetRel}{Retrieved}$$

and recall is defined as

$$R(Retrieved) = \frac{RetRel}{Relevant}.$$

We see that precision is the percentage of retrieved documents that are relevant to the query and recall is the percentage of relevant documents that have been retrieved.

1.2 Forming a Multidocument Summary

Another information retrieval problem is forming a multidocument summary. The formulation of this problem is similar to the problem of querying a document database; however, instead of a term-document matrix, we use a term-sentence matrix. In this case, each column of the term-sentence matrix corresponds to a sentence from a document included in the set to summarize.

One solution to this problem is to compute a dot product query score for each sentence in the term-sentence matrix and return the highest scoring sentences as those to be included in the multidocument summary. A secondary, time permitting objective of this project is to test the performance of approximate matrix decompositions in place of the term-sentence matrix in this process, where performance is measured in terms of how similar the computed summary is to human summaries.

Another existing solution to this problem is presented in [4]. In this solution a multidocument summary is built by first performing single document summaries using a hidden Markov model (HMM). The highest scoring sentences chosen by the HMM (for all documents in the set) are processed into a term-sentence matrix, which is then scaled and used in a pivoted QR algorithm. The results of this process are the sentences that should be included in the multidocument summary. (This is a general overview of the process. Other details that improve results are presented in [4].)

2 Approach

We plan to implement the following approximate matrix decompositions: an approximate nonnegative matrix factorization (NMF) computed by the multiplicative update algorithm of Lee and Seung as found in [1], and a linear time, Monte Carlo CUR decomposition by Drineas, Kannan, and Mahoney as found in [2].

We also plan to investigate and implement an improvement to the CUR algorithm found in [2]. The improvement is a compact matrix decomposition (CMD) by Sun, Xie, Zhang, and Faloutsos presented in [6]. We plan to investigate different sampling schemes for C and R and different methods of computing U as well. We hope these investigations will lead to improvements in storage, runtime, or relative error in the Frobenius norm of the CUR decomposition computed by the algorithm in [2]. (For the remainder of this paper, relative error refers to relative error in the Frobenius norm.)

After the above implementations are complete, we will investigate the storage, runtime, and relative error of the NMF and CUR decomposition. We will also test the performance of the NMF and CUR decomposition in LSI and investigate query time.

2.1 Approximate Nonnegative Matrix Factorization

In general, a term-document matrix is nonnegative; thus, it is an interesting problem to find an approximate nonnegative decomposition. Approximate nonnegative matrix factorization (NMF) as found in [1] does just this; an $m \times n$ nonnegative matrix A is decomposed as $A \approx WH$, where W is $m \times k$,

H is $k \times n$ and both W and H are nonnegative. For this NMF, k is a rank parameter. We use the multiplicative update algorithm of Lee and Seung as found in [1] to compute this factorization. The goal of this algorithm is to find a W and H such that

$$\frac{1}{2}\|A - WH\|_F^2$$

is minimized [1].

In this algorithm, W and H are randomly initialized. At each iteration H is updated using A and W and then W is updated using A and the new H. The updates are done by a series of multiplications and divisions [1].

Unfortunately, this algorithm does not have guaranteed convergence, and when it does converge, convergence can be slow; there are six matrix multiplications per iteration [1]. Fortunately, in practice, convergence is very common; also, slight modifications, such as grouping certain matrix multiplications, can speed up the runtime [1].

2.2 CUR Decomposition

The CUR decomposition is an approximation to an $m \times n$ matrix A as $A \approx CUR$, where C is $m \times c$, U is $c \times r$ and R is $r \times n$. The general idea behind this decomposition is as follows: C holds c sampled and rescaled columns of A, R holds r sampled and rescaled rows of A, and U is computed using C and R.

Another property common to term-document matrices is sparsity. The CUR decomposition preserves this sparsity property, whereas other decompositions such as the SVD, do not. This allows for not only less storage, but also a better physical interpretation of the decomposition; the basis vectors are (rescaled) document columns from the original matrix. Thus, each column from the product $CUR \approx A$, can be written as a linear combination of the columns of C.

2.2.1 Sampling

We investigate various sampling methods for the CUR decomposition:

- Column (Row) norm sampling with replacement [2]:
  $Prob(col\ j) = \|A(:,j)\|_F^2 / \|A\|_F^2$
  $Prob(row\ i) = \|A(i,:)\|_F^2 / \|A\|_F^2$

- Column (Row) norm sampling without replacement
- Subspace sampling with replacement [3]: Let the rank-k SVD of A be $A_k = U_k \Sigma_k V_k^T$ and the economy size SVD of C be $C = U_C \Sigma_C V_C^T$.
  $Prob(col\ j) = \|V_k(j,:)\|_F^2/k$
  $Prob(row\ i) = \|U_C(i,:)\|_F^2/c.$

2.2.2 Computing U

We also investigate two options for computing U for the CUR decomposition:

- Linear U [2]: approximately solves $\min_{\hat{U}} \|A - C\hat{U}\|_F$, where $\hat{U} = UR$, and $rank(U) \leq k$, where k is a rank parameter.
- Optimal U: solves $\min_U \|A - CUR\|_F$.

2.2.3 Implementations

We implement a CUR algorithm by Drineas, Kannan, and Mahoney as found in [2] that uses column (row) norm sampling and the linear U as explained above. However, this decomposition can be seen as an approximation to an approximation. Thus, the relative error can be far from optimal depending on the choices of c, r and k. Fortunately, this implementation is speedy; it runs in linear time [2].

We implement an improvement to the CUR algorithm as found in [2]. The improvement is a compact matrix decomposition (CMD) by Sun, Xie, Zhang, and Faloutsos presented in [6]. The CMD uses the same sampling and computation of U as the CUR algorithm of [2]. The CMD improvement is the following: remove repeated columns in C and repeated rows in R, then rescale the columns of C and rows of R appropriately, and then compute U. This improvement decreases storage space and runtime while achieving the same relative error [6].

Furthermore, we implement and compare the sampling and computation of U methods outlined above. We also compare with an implementation by G. W. Stewart of a deterministic CUR algorithm, which uses a rank revealing QR decomposition to determine which columns (rows) of A should be contained in C (R) and the optimal U computation.

3 Preliminary Results

In all preliminary results, we use a 50×30 random sparse test matrix that has rank 30. Call this matrix A. We can use A as a preliminary test matrix because the NMF and CUR decomposition can be used on any matrix. We chose A to be sparse because this is a property shared by a term-document matrix. Further, in all results, runtime is given in seconds, storage in number of nonzero entries, and relative error in the Frobenius norm. All implementations are done in MATLAB.

3.1 NMF

Our implementation of the multiplicative update algorithm of Lee and Seung as found in [1] provided the results seen in Figure 1. Due to the random initialization process, we take the average relative error and runtime over 5 runs for each value of k.

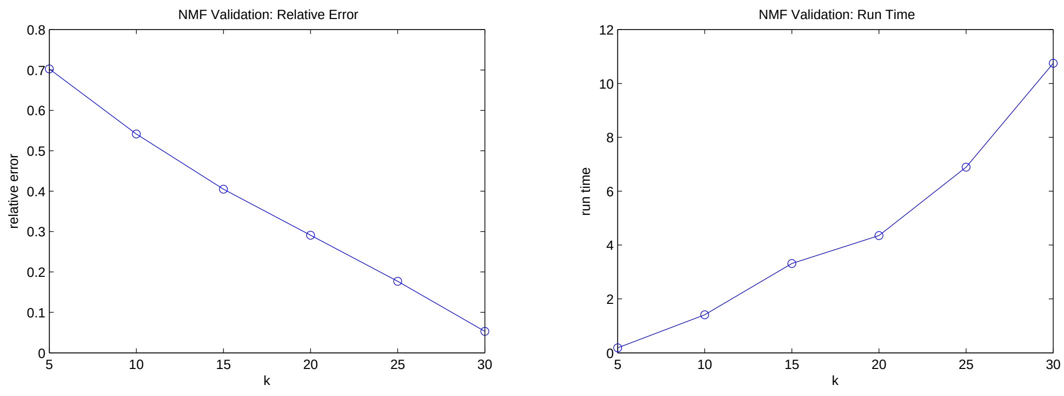

Figure 1: NMF preliminary results

We observe that as the rank parameter k increases to the rank of our test matrix A, the relative error of the NMF of A decreases towards zero; this is what we expect. However, when $k = 30$, we would like to see the relative error lower than it currently is, which is about 0.05. Also, we observe in Figure 1 that the NMF we implemented is slow. Future work includes determining if these results can be improved; one possible option is to implement an NMF using another algorithm, such as an alternating least squares algorithm, and see if an improvement is made in average relative error and/or speed.

3.2 CUR

We implemented the CUR algorithm as found in [2], and the CMD improvement to it as found in [6]. First, we present results in Table 1 that compare these two algorithms. Again, we use the test matrix A; we compute with $k = 15$, $c = r = 30$. We use the average runtime, storage, and relative error over 10 runs due to the Monte Carlo approach of the CUR algorithm in [2] and the CMD algorithm in [6].

Algorithm	[2]	[2] with CMD
Runtime	0.008060	0.007153
Storage	880.5	550.5
Relative Error	0.820035	0.820035

Table 1: CMD results

We see that the CMD achieves improvements in runtime and storage, while achieving the same relative error as the CUR algorithm as found in [2]; these results were expected [6]. The CMD decreases runtime because computations involving C with fewer columns are faster. In all CUR algorithms that use sampling with replacement, we have also implemented the CMD algorithm of [6].

Next, in Figure 2 we show a comparison of the CUR implementations using different sampling methods and different computations of U. We also include Stewart's deterministic CUR in the comparison. We use our test matrix A, and for the Monte Carlo CUR algorithms, we plot the average relative error and runtime for each value of k over 5 runs.

We use $c = r = 2k$ for all algorithms. However, for the algorithms which use the optimal U (including Stewart's deterministic CUR), we do not pass a rank parameter k; we do not control the rank of the optimal U. So, these algorithms are actually controlled by c and r, which depend on k.

The legend for the plots in Figure 2 corresponds to the following for each Monte Carlo CUR algorithm:

1. Sampling choice. CN : column (row) norm sampling with replacement [2], S: subspace sampling with replacement [3], w/o R: column (row) norm sampling without replacement

2. U choice. L: linear U [2], O: optimal U
3. Scaling (only for use with sampling without replacement). Sc: rescaling of columns and rows used, w/o Sc: rescaling of columns and rows not used.

Stewart's deterministic CUR is given in the legend by D.

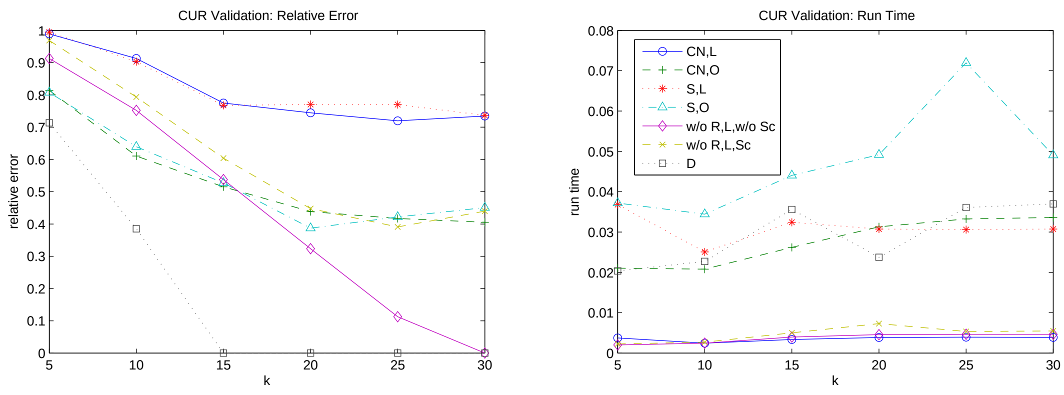

Figure 2: CUR preliminary results

We expect that as k increases, the relative error of the CUR decomposition of A should decrease. We see in Figure 2 that this general trend is followed by all CUR algorithms; however, we also expect that the relative error should be close to zero when $k = 30$. This is only accomplished by Stewart's deterministic CUR algorithm and the CUR algorithm that samples without replacement with respect to column (row) norm probabilities, uses the linear U, and does not rescale columns (rows) in C (R).

In fact, Stewart's CUR algorithm has a relative error of approximately zero when $k = 15$; this is expected. The rank of A is 30 and when $k = 15$, $c = r = 30$, giving us the dimensions 30×30 for U, which is computed optimally.

Other general trends we see from Figure 2 for our test matrix A are:

- The optimal U is more expensive than the linear U, but gives better relative error results.
- Subspace sampling and column (row) norm sampling (both with replacement) give similar relative error results, although subspace sampling is more expensive.

- Sampling without replacement with respect to column (norm) probabilities using the linear U gives improvements in relative error over sampling with replacement with respect to column (norm) probabilities using the linear U while remaining inexpensive.

In practice it appears we may have found a Monte Carlo CUR algorithm that improves results produced by the CUR algorithm from [2] while remaining inexpensive; namely the CUR algorithm that samples without replacement with respect to column (row) norm probabilities, uses the linear U, and does not rescale columns (rows) in C (R).

Future work includes trying to determine why rescaling or not rescaling columns (rows) in C (R) has the effect seen in Figure 2 when using sampling without replacement with respect to column (row) norm probabilities and the linear U.

4 Further Work

Further work for this project includes finalizing the NMF and CUR algorithms as explained above. Also, testing and validation of the NMF and CUR algorithms will be continued using other matrices, specifically three term-document matrices that are common information retrieval databases. These three matrices can be found at www.cs.utk.edu/~lsi/ and are listed under CISI, CRAN, and MED.

Once validation of the NMF and CUR decomposition is complete, we will investigate storage, runtime, and relative error for both the NMF and CUR decomposition. Then, we will test the performance of the NMF and CUR decomposition in LSI.

Let A be an $m \times n$ term-document matrix, q an $m \times 1$ query vector, and s an $1 \times n$ score vector such that s_i is the relevance score for document i for the given query. In LSI, for a given query, we will compute s using a series of vector-matrix products as follows:

- (rank-k) SVD: $A \approx U_k \Sigma_k V_k^T$, $s = ((q^T U_k)\Sigma_k)V_k^T$
- NMF: $A \approx WH$, $s = (q^T W)H$
- CUR: $A \approx CUR$, $s = ((q^T C)U)R$.

We will test the performance of the NMF and CUR decomposition in LSI using average precision and recall, where the average is taken over all queries in the data set, and compare to the performance of the SVD in LSI. We will also compare query time for the NMF, CUR decomposition, and SVD. We will complete this process for each of the three term-document matrices referenced above.

5 Time Permitting Investigations

Time permitting, we will test the performance of the NMF and CUR decomposition as the approximation to the term-sentence matrix in multidocument summarizing. We would use dot product query scores to score sentences for relevance. Furthermore, we would use a data package from John Conroy of the Center for Computing Sciences, Institute for Defense Analysis. The package would include term-sentence matrices and query vectors.

To validate our generated multidocument summaries we would use ROUGE, the Recall-Oriented Understudy for Gisting Evaluation, to score them against human summaries. A ROUGE score is a number between 0 and 1 that gives a comparison of two summaries: the more similar the summaries, the higher the ROUGE score [4].

Due to the fact that computations with large matrix data can be costly, another time permitting investigation is to code parallel implementations of the NMF and CUR decomposition.

6 Project Schedule

January: Compile databases in an organized and easily accessible manner, continue and finish NMF and CUR investigations, check NMF and CUR decomposition for efficiency (if time, investigate parallelization)

February: Test and validate NMF and CUR decomposition using term-document matrices, analyze NMF and CUR decomposition for speed, storage and relative error

March: Test and validate use of NMF and CUR decomposition in LSI, analyze query time in LSI using NMF and CUR decomposition, (if time, test and validate use of NMF and CUR decomposition in forming multidocument summaries)

April: Write final report

May: Present final report

Deliverables: Code, final report

References

[1] Michael W. Berry, Murray Browne, Amy N. Langville, V. Paul Pauca, and Robert J. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Computational Statistics and Data Analysis*, 52(1):155–173, September 2007.

[2] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast Monte Carlo algorithms for matrices iii: Computing a compressed approximate matrix decomposition. *SIAM Journal on Computing*, 36(1):184–206, 2006.

[3] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Relative-error CUR matrix decompositions. *SIAM Journal on Matrix Analysis and Applications*, 30(2):844–881, 2008.

[4] Daniel M. Dunlavy, Dianne P. O'Leary, John M. Conroy, and Judith D. Schlesinger. QCS: A system for querying, clustering and summarizing documents. *Information Processing and Management*, 43(6):1588–1605, November 2007.

[5] Tamara G. Kolda and Dianne P. O'Leary. A semidiscrete matrix decomposition for latent semantic indexing in information retrieval. *ACM Transactions on Information Systems*, 16(4):322–346, October 1998.

[6] Jimeng Sun, Yinglian Xie, Hui Zhang, and Christos Faloutsos. Less is more: Sparse graph mining with compact matrix decomposition. *Statistical Analysis and Data Mining*, 1(1):6–22, February 2008.