# 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 [4].

### 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 [4].

### 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 [4] 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 [4].

In [4], 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 [3]. 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 [3].)

# 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 algorithm by Drineas, Kannan, and Mahoney as found in [2].

We also plan to investigate and implement an improvement to this CUR algorithm. The improvement is a compact matrix decomposition (CMD) by Sun, Xie, Zhang, and Faloutsos presented in [5]. 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 of the CUR matrix approximation computed by the algorithm mentioned above.

After the above implementations are complete, we will investigate the storage, runtime, and relative error of each approximate matrix decomposition. We will also test the performance of each approximate matrix 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].

Unfortunately, this algorithm does not have guaranteed convergence, and when it does converge, convergence can be slow [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 $R$.

We implement a CUR algorithm by Drineas, Kannan, and Mahoney as found in [2] that approximately solves the following least squares problem:

$$\min_{\hat{U}} \|A - C\hat{U}\|_F,$$

where $\hat{U}$ is $c \times n$, $\hat{U} = UR$, and $rank(U) \leq k$, where $k$ is a rank parameter. However, this decomposition can be seen as an approximation to an approximation. Thus, the relative error (in the Frobenius norm) 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 also implement an improvement to this CUR algorithm. The improvement is a compact matrix decomposition (CMD) by Sun, Xie, Zhang, and Faloutsos presented in [5]. The 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 [5].

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 $C$.

# 3 Implementation

All implementations will be done in MATLAB. Due to the potential for matrix data with very large dimensions, the computations performed may become costly. Thus, time permitting, parallel implementation may be investigated. Information retrieval is a common task for all computers; thus, we plan to run this software on a PC.

# 4 Databases

We use three different 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. Query vectors and a list a relevant documents for each query are also available for each of these matrices at the same website. Thus, each database will consist of a term-document matrix, query vectors, and a list of relevant documents for each query.

# 5 Validation and Testing

We will validate each approximate matrix decomposition using the following process: (let $\widetilde{A}$ be an approximate matrix decomposition of an $m \times n$ matrix $A$) we compute $\widetilde{A}$ multiple times using a different value of the rank parameter $k$ each time. We verify that as $k$ goes to $\min(m, n)$, the relative error of $\widetilde{A}$ goes to zero. This will confirm that $\widetilde{A}$ is in fact accurately decomposing $A$.

We will validate our document retrieval results by comparing with results available at www.cs.utk.edu/~lsi/, which consist of a list of relevant documents for each query. We will compute average values of precision and recall, where the average is taken over all queries in the data set, and compare to the performance achieved by LSI using the SVD.

We will complete these processes for each of the three databases referenced above. Additional databases could be created using other document

collections found at www.cs.utk.edu/~lsi/ and tested in this project using the same procedure outlined above.

# 6 Time Permitting Investigation

Time permitting, we will test the performance of each approximate matrix 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 [3].

# 7 Project Schedule, Milestones, and Deliverables

**October**: Investigate and implement existing approximate decompositions, write and present project proposal

**November**: Implement and analyze CMD improvement for CUR decomposition as found in [5], investigate other changes/improvements to the implemented CUR decomposition

**December**: Continue to investigate changes/improvements to the implemented CUR decomposition, write and present midterm report

**January**: Compile databases in an organized and easily accessible manner, continue and finish CUR investigation, check all implemented approximate decompositions for efficiency (if time, investigate parallelization)

**February**: Test and validate implemented approximate decompositions, analyze implemented approximate decompositions for speed, storage and rela-

tive error

**March**: Test and validate use of implemented approximate decompositions in LSI, analyze query time in LSI using implemented approximate decompositions, (if time, test and validate use of implemented approximate decompositions 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] 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.

[4] 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.

[5] 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.