Automated Gene Classification using Nonnegative Matrix Factorization on Biomedical Literature

A Dissertation

Presented for the

Doctor of Philosophy Degree

The University of Tennessee, Knoxville

Kevin Erich Heinrich

May 2007

Acknowledgments

I would like to express my deep gratitude to Dr. Michael Berry for the support, guidance, and encouragement given to me through the high and low points of my academic career. From way over in Memphis, Dr. Ramin Homayouni at the University of Memphis provided excitement and energy that helped motivate me over the years, and for that I am extremely grateful. If not for these two, this dissertation would have never come to fruition. I would like to thank Drs. Jens Gregor and Michael Thomason for supporting my doctoral candidacy by serving on my committee, and Dr. Paúl Pauca at Wake Forest University for helping me better understand NMF. Most importantly, I would like to express my love and appreciation for my family and my wife, Erica, for believing in me and putting up with me while I was completing my work.

This work was supported by the Center for Information Technology Research and the Science Alliance Computational Sciences Initiative at the University of Tennessee and by the National Institutes of Health under Grant No. HD52472-01.

Abstract

Understanding functional gene relationships is a challenging problem for biological applications. High-throughput technologies such as DNA microarrays have inundated biologists with a wealth of information, however, processing that information remains problematic. To help with this problem, researchers have begun applying text mining techniques to the biological literature. This work extends previous work based on Latent Semantic Indexing (LSI) by examining Nonnegative Matrix Factorization (NMF). Whereas LSI incorporates the singular value decomposition (SVD) to approximate data in a dense, mixed-sign space, NMF produces a parts-based factorization that is directly interpretable. This space can, in theory, be used to augment existing ontologies and annotations by identifying themes within the literature. Of course, performing NMF does not come without a price—namely, the large number of parameters. This work attempts to analyze the effects of some of the NMF parameters on both convergence and labeling accuracy. Since there is a dearth of automated label evaluation techniques as well as "gold standard" hierarchies, a method to produce "correct" trees is proposed as well as a technique to label trees and to evaluate those labels.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The emergence of high-throughput techniques in genomics, proteomics, and related biological fields have generated large amounts of data that enable researchers to examine biological systems from a global viewpoint. Unfortunately, however, the sheer mass of information available is overwhelming, and data such as gene expression profiles from DNA microarray analysis can be difficult to understand fully even for domain experts. Additionally, performing these experiments in the lab can be expensive with respect to both time and money.

With access to online repositories of biomedical literature becoming widely available in recent years, text mining techniques have been applied to these databases and are becoming increasingly accepted as a method to validate lab results and to direct future research by uncovering latent relationships. [JLKH01] was among the first to utilize online text to expose gene relationships by exploiting term co-occurrence patterns within the literature.

In many cases, the information gleaned from online databases is related to gene function.[1] In the biological context, uncovering gene function (phenotype) is one of the primary goals of research and is also one of the most difficult to verify. In general, protein structure, not the primary sequence (genotype), indicates function. Unfortunately, structure analysis is expensive, time-consuming, and difficult—current techniques may not even reveal true structure as proteins must be first crystallized to be examined. Protein function, however, can be examined biochemically, although many of the available methods remain remain time-consuming and expensive.

Since direct examination of proteins is difficult, literature has become an accepted data source to examine phenotype. Unfortunately, many of the literature sources are themselves imperfect; many of the medical literature databases online are manually indexed. While manual annotation adds a degree of confidence to an association, the subjectivity involved can be scrutinized. For example, [FR83] has shown that different manual indexers of MEDLINE produce consistent associations only 40–50% of the time. Given the time required to read and classify an article, automated methods may help increase the annotation rate as well as improve existing annotations.

One such tool that may help improve annotation as well as identify functional groups of genes is the Semantic Gene Organizer (SGO). SGO is a software environment based upon latent semantic indexing (LSI) that enables researchers to view groups of genes in a global context as a hierarchical tree or dendrogram [Hei04]. The low-rank approximation pro-

[1] More specifically, genes code for proteins which in turn carry out a specific function.

vided by LSI (for the original term-to-document associations) exposes latent relationships between genes, and the resulting hierarchical tree is a visualization of those relationships that is reproducible and easily interpreted by biologists; [HHWB05] has shown that the SGO can identify groups of related genes more accurately than term co-occurrence methods. LSI is based upon the singular value decomposition (SVD) [GL96], and the properties of the SVD are well-known. Unfortunately, since the input data is a nonnegative matrix of weighted term frequencies, the negative values prevalent in the basis vectors of the SVD are not easily interpreted.

The decomposition produced by the recently popular nonnegative matrix factorization (NMF), however, can be readily interpreted. [PT94] is among the first in recent years to investigate this factorization, and [LS99] later popularized it. Generally speaking, NMF is an iterative algorithm that preserves the nonnegativity of the original data. The factorization produces a low-rank, parts-based representation of the original data. In effect, common themes present in the data can be identified simply by inspecting the factor matrices. Depending on the interpretation, the factorization can induce both clustering and classification. If NMF can accurately model the input data, then it can be used to classify that data. Within the context of SGO, this means that the groups of genes presented in the hierarchical trees can be assigned labels that identify their common attributes such as protein function.

The interpretability of NMF, however, comes at a price. Namely, convergence and stability are not guaranteed. Also, many variations of the NMF proposed by [LS99] exist,

each with different parameters. The goals of this study are twofold: to provide a qualitative assessment of the NMF and its various parameters, particularly as they apply to the biomedical context, and to provide an automated way to classify biomedical data as well as provide a method for evaluating that labeled data assuming a static input tree.[2] As a byproduct, a method for generating "gold standard" trees is proposed.

To appreciate how NMF is applied in this context, a general understanding of SGO is required. First, a background chapter describing the vector space model and gene document construction process employed by SGO is presented in Chapter 2. Chapter 3 discusses the two factorization schemes in detail—both the SVD and NMF. Tree construction and labeling algorithms are examined in Chapter 4. Finally, Chapter 5 evalutes the performance of NMF, and Chapter 6 discusses future research.

[2]The input tree is assumed to be static to avoid biological implications of changing known associations.

Chapter 2

The Vector Space Model

Traditionally, the field of information retrieval (IR) is divided into Boolean, probabilistic, and vector space models [BYRN99]. Within the context of SGO, knowledge of the general vector space model and the impact of term weighting is necessary. Before the documents can be constructed, however, an understanding of the document creation process is useful; unlike most IR applications, document construction is a non-trivial process within the SGO environment.

2.1 Data Representation

An underlying assumption of the vector space model is that meaning can be extracted from a document solely based on the words that comprise it. After removing articles and other commonly used terms (*stopwords*) from the text and ignoring capitalization and most punctuation, the remaining non-alphanumeric terms (atomic units called *tokens*) comprise

a parsed document; all tokens within a corpus form a dictionary.

Further preprocessing steps can be performed on the dictionary to limit the vocabulary to the most significant terms. For example, many terms can be excluded from the dictionary if they do not meet a certain frequency threshold either globally (within the corpus) or locally (within the document), or similarly if they exceed a threshold.

Another common preprocessing procedure is applying a *stemming* algorithm such as the Porter Stemmer [vRRP80, Por80]. This algorithm attempts to stem all forms of a word into its common base, essentially stripping all prefixes and suffixes from a word in an attempt to eliminate all morphological variation. For example, "weighting" and "weights" stem back to the base word "weight." Determining the base "word" to stem to is, unfortunately, a difficult and imperfect process that may introduce additional errors. Overall, however, stemming is understood to have a marginal impact on retrieval results [Hul96].

While stemming is a way to reduce the vocabulary size, *windowing* is a way to change vocabulary size, usually by increasing it [Dam95]. By using a sliding window of size n across the text, each token becomes an n-gram of characters. Granted much overlap is produced for each word, the dictionary is guaranteed to be at most size Σ^n, where Σ is the number of characters in the language (usually plus the blankspace). Although the effectiveness of this approach can be argued [HBC+95], the idea of a sliding window has merit in some application areas. For example, by extending the sliding window's atomic unit to words instead of characters, phrases can be indexed (e.g. "New York City" can be indexed as one token rather than three separate ones). In the field of bioinformatics, [SB03]

has applied the sliding window concept to proteins to develop *peptide motifs* that comprise the dictionary. Regardless of the preprocessing step used, for the vector space model to be successful, the dictionary must contain most if not all of the tokens (after stemming, windowing, or another procedure is applied) that will be used to query.

Once the dictionary is built, a corpus is represented by an $m \times n$ *term-by-document matrix* $A = [w_{ij}]$, where w_{ij} is the weight associated with token i in document j, m is the number of tokens in the dictionary, and n is the number of documents in the corpus. Furthermore, the columns of A correspond to document vectors of length m, while the rows of A are term vectors of length n. In this particular context, the document vectors or columns of A also correspond to particular genes. A sample corpus and corresponding term-document matrix is given in Tables 2.1 and 2.2; the log-entropy weighting scheme discussed in Section 2.2 was used to generate the term weights.

2.2 Term Weighting

Defining the appropriate weighting scheme for each corpus is application-dependent. Typically, each weight w_{ij} is based on some function of the frequency of a token within a document or the entire corpus. In general,

$$w_{ij} = l_{ij} g_i d_j,$$

Table 2.1: Sample collection with dictionary terms displayed in **bold**.

Document	Text
d1	Work-related **stress** can be considered a factor contributing to **anxiety**.
d2	**Liver cancer** is most commonly associated with **alcoholism** and **cirrhosis**. It is well-known that **alcoholism** can cause **cirrhosis** and increase the risk of **kidney failure**.
d3	**Bone marrow** transplants are often needed for patients with **leukemia** and other types of **cancer** that **damage bone marrow**. Exposure to toxic chemicals is a risk factor for **leukemia**.
d4	Different types of **blood cells** exist in **bone marrow**. **Bone marrow** procedures can detect **tuberculosis**.
d5	Abnormal **stress** or **pressure** can cause an **anxiety attack**. Continued **stress** can elevate **blood pressure**.
d6	**Alcoholism** can cause high **blood pressure** (**hypertension**) and increase the risk of **birth defects** and **kidney failure**.
d7	The presence of **speech defects** in **children** is a sign of **autism**. As of yet, there is no consensus on what causes **autism**.
d8	**Alcoholism**, often triggered at an early age by factors such as environment and genetic predisposition, can lead to **cirrhosis**. **Cirrhosis** is the **scarring** of the **liver**.
d9	**Autism** affects approximately 0.5% of **children** in the US. The link between **alcoholism** and **birth defects** is well-known; researchers are currently studying the link between **alcoholism** and **autism**.

Table 2.2: Term-document matrix for the sample collection in Table 2.1.

	d1	d2	d3	d4	d5	d6	d7	d8	d9
alcoholism		0.4338				0.2737		0.2737	0.4338
anxiety	0.4745				0.4745				
attack					0.6931				
autism							0.7520		0.7520
birth						0.4745			0.4745
blood				0.3466	0.3466	0.3466			
bone			0.7520	0.7520					
cancer		0.4745	0.4745						
cells				0.6931					
children							0.4745		0.4745
cirrhosis		0.7520						0.7520	
damage			0.6931						
defects						0.3466	0.3466		0.3466
failure		0.4745				0.4745			
hypertension						0.6931			
kidney		0.4745				0.4745			
leukemia			1.0986						
liver		0.4745						0.4745	
marrow			0.7520	0.7520					
pressure					0.7804	0.4923			
scarring								0.6931	
speech							0.6931		
stress	0.4923				0.7804				
tuberculosis				0.6931					

where l_{ij} denotes the local weight of term i in document j, g_i corresponds to the global weight of term i, and d_j is a document normalization factor for the columns of A. Since most weighting schemes are based on word frequency, using a normalization factor helps eliminate the effect of document length discrepancies and spamming.

By default, SGO utilizes the *log-entropy* weighting scheme, given by

$$\begin{aligned} l_{ij} &= \log(1 + f_{ij}), \\ g_i &= 1 + \left(\frac{\sum_j (p_{ij} \log p_{ij})}{\log n} \right), \end{aligned}$$

where f_{ij} is the frequency of token i in document j and $p_{ij} = f_{ij} / \sum_j f_{ij}$ is the probability of token i occurring in document j. By design, tokens that appear less frequently across the collection but more frequently within a document will be given higher weight. That is, distinguishing tokens will tend to have higher weights assigned to them, while more common tokens will have weights closer to zero.

2.3 Similarity

Measuring similiarity between any two vectors in the document space is accomplished by computing the cosine of the angle between them and is represented by

$$sim(d_i, d_j) = \cos\Theta_{ij} = \frac{\sum_{k=1}^{m} w_{ki} w_{kj}}{\sqrt{\sum_{k=1}^{m} w_{ki}^2} \sqrt{\sum_{k=1}^{m} w_{kj}^2}},$$

where Θ_{ij} is the angle between any two documents i and j. In the vector space model, nonzero similarities exist only if two documents share common terms; this caveat is evidenced by the sample collection similarity matrix presented in Table 2.3.

On the other hand, a measure of distance can be calculated by

$$dist\,(d_i, d_j) = 1 - \cos\Theta_{ij}. \tag{2.1}$$

The numerical similarity and distance values by themselves carry little meaning; however, such measures are useful when constructing similarity and distance matrices that show pairwise similarities and distances between every combination of two documents in the collection. In fact, this measure is not a metric; that is, Equation (2.1) does not satisfy the triangle inequality (or *addivity*) required by many tree construction algorithms to guarantee the production of a "correct" tree. However, the "best fit" tree can be found by minimizing Equation (4.1).

2.4 Gene Document Construction

An abundance of medical and biological literature exists in databases worldwide. The United States National Library of Medicine® (NLM) maintains one such bibliographic database, called MEDLINE® (Medical Literature, Analysis, and Retrieval System Online). MEDLINE covers topics that concern biomedical practitioners such as biomedical research, chemical sciences, clinical sciences, and life sciences. Currently, MEDLINE con-

Table 2.3: Document-to-document similarity matrix for the sample collection.

	d1	d2	d3	d4	d5	d6	d7	d8	d9
d1	1	0	0	0	0.6234	0	0	0	0
d2	0	1	0.1003	0	0	0.3380	0	0.6094	0.1271
d3	0	0.1003	1	0.4359	0	0	0	0	0
d4	0	0	0.4359	1	0.0565	0.0617	0	0	0
d5	0.6234	0	0	0.0565	1	0.2695	0	0	0
d6	0	0.3380	0	0.0617	0.2695	1	0.0778	0.0493	0.3081
d7	0	0	0	0	0	0.0778	1	0	0.6711
d8	0	0.6094	0	0	0	0.0493	0	1	0.0889
d9	0	0.1271	0	0	0	0.3081	0.6711	0.0889	1

tains over 15 million citations to biologically-related articles spanning over 5,000 journals, with approximately 2,000–4,000 citations added daily [NLM]. Each citation is manually tagged with Medical Subject Heading® (MeSH) terms; MeSH is a controlled vocabulary provided by NLM organized in a hierarchical fashion.

NLM provides PubMed and NLM Gateway as tools to search MEDLINE. Since MEDLINE covers a broad range of topics and genes themselves have aliases, simply querying PubMed for a particular gene is not guaranteed to return citations relevant to the gene in question; PubMed searches are subject to the problems of synonymy and polysemy. For a more gene-centric search, LocusLink can be used. LocusLink is provided by the National Center for Biotechnology Information (NCBI), a division of NLM that specializes in molecular biology. LocusLink, which has been upgraded to EntrezGene [MOPT05], is a human-curated database that contains a myriad of information pertaining to each gene. In particular, many genes are linked to key relevant MEDLINE citations via PubMed. Unfortunately, since EntrezGene is human-curated, its coverage of MEDLINE and other databases is sparse, but its lack of quantity of citations is compensated by the quality of its

citations.

Unlike traditional IR fields, creating a document collection is an iterative, non-trivial process requiring expert knowledge. To create a *gene document* (text document to represent a gene), an ad hoc method is applied [Hei04]. Titles and abstracts of all MEDLINE citations generated via PubMed cross-referenced in the Mouse, Rat, and Human EntrezGene (or LocusLink) entries are concatenated into a document for each gene. Since the coverage of MEDLINE provided by EntrezGene and PubMed is both incomplete and imperfect, many errors can be introduced. For example, many of the genes have aliases since their names have changed over time or depending on which field is referencing them. As such, trying to find all references to a particular gene can be difficult. Filtering techniques iteratively applied to the gene document collection can produce more accurate gene representations and are discussed in [HHWB05]. After the process is complete, each gene document may not have complete coverage of all related MEDLINE material, but each of its constituent abstracts will have been validated at some level by manual curation, hopefully resulting in a good representative document.

2.5 MeSH Meta-Collection

In addition to the constructing gene documents, constructing MeSH documents can help with the evaluation process. Since each abstract associated with a gene is tagged with possibly multiple MeSH headings by a human indexer, a MeSH gene document can be created by concatenating those headings. In effect, this approach takes advantage of known

associations indicated by the human indexer to create a meta-collection of MeSH terms that can help summarize the information contained in the original gene corpus. Although MeSH is a hierarchy, that hierarchy is ignored when headings are assigned to an abstract. Also, the MeSH headings can themselves be weighted with log-entropy weights to help find more distinguishing headings. These weighted MeSH headings can be assigned to a hierarchical tree via the method outlined in Algorithm 4.1, where each node is represented by its dominant MeSH heading. The resulting labeling can be considered a "gold standard" against which to compare.[1] For the purposes of this dissertation, the MeSH meta-collection constructed for a collection is indicated by appending "_MeSH" to the collection name (e.g., if the *50TG* collection is being examined, then *50TG_MeSH* indicates the MeSH meta-collection associated with *50TG*).

[1]This procedure is explained in more detail is Section 4.2.

Chapter 3

Dimension Reduction Techniques

The vector space model presented in Chapter 2 suffers from the *curse of dimensionality*. In other words, as the problem sizes increase and become more complex, the processing time required to construct a vector space and query throughout the document space increases as well. In addition, the vector space model exclusively measures term co-occurence—that is, the inner product between two documents is nonzero if and only if there exist at least one shared term between them. No attempt to discern the underlying structure of the data is made, and problems such as *synonymy*[1] and *polysemy*[2] will reduce the effectiveness of the model.

To combat the problem of dimensionality, dimension reduction techniques are typically employed. A multitude of techniques exist, each with its own strengths and weaknesses. To combat the synonymy problem, Latent Semantic Indexing (LSI) is discussed. Nonnegative

[1] If two different words have the same meaning, then they are synonyms.
[2] Polysemy refers to words having multiple definitions.

Matrix Factorization (NMF) is examined to analyze further the underlying structure of the data set in question.

3.1 Latent Semantic Indexing

LSI is based on the singular value decomposition (SVD). Given a matrix, A, the SVD is given by

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

where U is the $m \times r$ matrix of eigenvectors of AA^T, Σ is the $r \times r$ diagonal matrix of r singular values of A, V^T is the $r \times n$ matrix of eigenvectors of $A^T A$, and r is the rank of A. Furthermore, both U and V have orthonormal columns, and the diagonal matrix Σ has, by convention, only positive elements ordered by decreasing magnitude [GL96]. If an SVD is performed on the sample term-document matrix, the left singular matrix is given by

$$U = \begin{pmatrix}
0.0563 & -0.3569 & -0.1740 & 0.0172 & -0.0333 & 0.0203 & 0.0878 & -0.3844 & 0.0897 \\
0.0189 & -0.1156 & 0.3191 & 0.0723 & 0.1516 & -0.1855 & -0.0951 & -0.1927 & -0.5269 \\
0.0234 & -0.1356 & 0.3521 & 0.0778 & 0.1355 & -0.1565 & -0.0040 & 0.0441 & 0.5135 \\
0.0200 & -0.2783 & -0.1083 & -0.5975 & 0.1165 & -0.2264 & -0.0296 & -0.1097 & 0.0917 \\
0.0306 & -0.2523 & 0.0096 & -0.1781 & -0.0957 & 0.2573 & 0.3105 & -0.3327 & -0.0403 \\
0.1270 & -0.1586 & 0.2396 & 0.0209 & -0.2300 & 0.0327 & 0.0593 & 0.1308 & 0.1034 \\
0.5231 & 0.1042 & 0.0206 & -0.0412 & -0.1411 & -0.0705 & 0.0180 & -0.0075 & -0.0201 \\
0.2261 & -0.0938 & -0.1417 & 0.1206 & 0.2138 & 0.0757 & -0.3714 & -0.1106 & 0.1234 \\
0.1980 & 0.0352 & 0.0566 & -0.0311 & -0.4297 & -0.1915 & -0.0695 & -0.0409 & -0.0051 \\
0.0126 & -0.1758 & -0.0684 & -0.3774 & 0.0736 & -0.1430 & -0.0186 & -0.0692 & 0.0579 \\
0.0692 & -0.3585 & -0.3502 & 0.3751 & 0.0360 & -0.3172 & -0.0176 & 0.0666 & 0.0337 \\
0.2841 & 0.0609 & -0.0375 & -0.0068 & 0.2996 & 0.1265 & 0.0862 & 0.0340 & -0.0133 \\
0.0254 & -0.2367 & -0.0149 & -0.2781 & -0.0289 & 0.1025 & 0.0825 & 0.0786 & -0.1088 \\
0.0535 & -0.2838 & -0.0680 & 0.1220 & -0.1047 & 0.2724 & -0.2987 & 0.0429 & -0.0742 \\
0.0323 & -0.2162 & 0.0699 & -0.0048 & -0.1651 & 0.4135 & 0.1922 & 0.2581 & -0.3018 \\
0.0535 & -0.2838 & -0.0680 & 0.1220 & -0.1047 & 0.2724 & -0.2987 & 0.0429 & -0.0742 \\
0.4506 & 0.0965 & -0.0595 & -0.0108 & 0.4752 & 0.2006 & 0.1367 & 0.0539 & -0.0211 \\
0.0437 & -0.2264 & -0.2212 & 0.2369 & 0.0228 & -0.2003 & -0.0111 & 0.0421 & 0.0213 \\
0.5231 & 0.1042 & 0.0206 & -0.0412 & -0.1411 & -0.0705 & 0.0180 & -0.0075 & -0.0201 \\
0.0492 & -0.3062 & 0.4459 & 0.0841 & 0.0352 & 0.1173 & 0.1318 & 0.2329 & 0.3636 \\
0.0181 & -0.1324 & -0.1535 & 0.1628 & 0.0209 & -0.2762 & 0.6119 & 0.2569 & -0.1622 \\
0.0061 & -0.1045 & -0.0439 & -0.2956 & 0.0820 & -0.1705 & -0.2880 & 0.6424 & -0.1583 \\
0.0293 & -0.1762 & 0.4768 & 0.1072 & 0.2133 & -0.2572 & -0.1002 & -0.1813 & -0.3323 \\
0.1980 & 0.0352 & 0.0566 & -0.0311 & -0.4297 & -0.1915 & -0.0695 & -0.0409 & -0.0051
\end{pmatrix},$$

the right singular matrix by

$$V = \begin{pmatrix} 0.0119 & -0.0822 & 0.2514 & 0.0590 & 0.1479 & -0.1880 & -0.1354 & -0.2914 & -0.8750 \\ 0.1299 & -0.4916 & -0.3751 & 0.3896 & 0.0212 & -0.0264 & -0.6321 & -0.1750 & 0.1320 \\ 0.8064 & 0.1512 & -0.0832 & -0.0146 & 0.5174 & 0.2084 & 0.0867 & 0.0305 & -0.0090 \\ 0.5619 & 0.0873 & 0.1253 & -0.0664 & -0.7420 & -0.3155 & -0.0699 & -0.0366 & -0.0035 \\ 0.0663 & -0.3371 & 0.7802 & 0.1657 & 0.2340 & -0.2579 & -0.0041 & 0.0395 & 0.3504 \\ 0.0916 & -0.5373 & 0.1548 & -0.0103 & -0.2851 & 0.6812 & 0.1934 & 0.2311 & -0.2059 \\ 0.0173 & -0.2599 & -0.0974 & -0.6299 & 0.1415 & -0.2810 & -0.2899 & 0.5753 & -0.1080 \\ 0.0512 & -0.3292 & -0.3402 & 0.3468 & 0.0361 & -0.4551 & 0.6158 & 0.2300 & -0.1107 \\ 0.0350 & -0.3774 & -0.1238 & -0.5434 & 0.0439 & -0.0627 & 0.2625 & -0.6657 & 0.1658 \end{pmatrix},$$

and the singular values on the diagonal of Σ by

$$\text{diag}(\Sigma) = (1.967, 1.7217, 1.5358, 1.4765, 1.1965, 1.1417, 0.6974, 0.6205, 0.4729).$$

To generate a rank-s approximation A_s of A where $s < r$, each matrix factor is truncated to its first s columns. That is, A_s is computed as

$$A_s = U_s \Sigma_s V_s^T.$$

Not only is A_s a lower rank approximation of A, it is the best rank-s approximation of A with respect to the Frobenius norm; [EY36] has shown that

$$\|A - A_s\|_F \leq \|A - B\|_F$$

for any matrix B of rank s. In this lower rank space, document-to-document similarity is computed as the inner product of the two documents in question. In matrix form,

$$A_s^T A_s = (V_s \Sigma_s)(V_s \Sigma_s)^T$$

will denote the similarity of each document in the corpus to every other. This calculation measures the dot product between the columns of A_s.

Similarly, term-to-term similarity can be computed as the inner product of the two terms in question. In matrix form,

$$A_s A_s^T = (U_s \Sigma_s) (U_s \Sigma_s)^T$$

yields a square, symmetric matrix that contains the dot product between any two rows in the scaled termspace.

One of the main drawbacks of the standard vector space model is that there is no straightforward way to compare terms to documents should the need arise. Since A is $m \times n$ where m differs from n most of the time, a dot product cannot be taken between document and term vectors. In the rare case where m equals n, the difference in the definition of axis components would make such a comparison meaningless. With a low-rank, square matrix approximation A_s, however, comparing a document with a term is possible.

Rather than performing a dot product of rows or columns of A_s, inspection of the definition of A_s yields a similarity value between each term/document pair. That is, given term i and document j, and by observing that

$$A_s = U_s \Sigma_s V_s^T, \tag{3.1}$$

the dot product between the ith row of $U_s \Sigma^{1/2}$ and the jth column of $V_s \Sigma^{1/2}$ represents the

similarity between term i and document j. In other words, the factored term and document axes must be stretched or shrunk by a factor of $\Sigma^{1/2}$ [DDL+90].

If document-to-document similarity is computed in the rank-4 space provided by the SVD for the sample collection, inspection of the matrix in Table 3.1 shows that intuitive document similarities are present. Although documents d1 and d6 share no common terms, $sim(\text{d1}, \text{d6}) = 0.55$, probably since *pressure* is related to *stress* via document d5. This comparison is made because the lower-rank space provided by the SVD forces documents to be represented by fewer components. Hence, hints of the power of LSI as well as some of the possible shortcomings of the bag of words model are present (since *pressure* can be considered to be used in two different contexts in documents d5 and d6).

3.2 Nonnegative Matrix Factorization

In its simplest form, given an $m \times n$ nonnegative matrix A, nonnegative matrix factorization (NMF) is a technique that attempts to find two nonnegative factor matrices, W and H, such that

$$A \approx WH, \tag{3.2}$$

where W and H are $m \times k$ and $k \times n$ matrices, respectively. In the context of this dissertation, $A = [a_{ij}]$ is an $m \times n$ matrix that represents the gene document collection. Each entry a_{ij} represents the term weight[3] of token i in gene document j. Hence, the rows of

[3]Section 2.2 discusses term weighting in more detail.

Table 3.1: Document-to-document similarity matrix for the sample collection. Similarities were calculated by computing the cosine between each normalized document pair in the rank-4 space generated by the SVD.

	d1	d2	d3	d4	d5	d6	d7	d8	d9
d1	1	-0.0918	-0.0752	0.1469	0.9941	0.5515	-0.1698	-0.1901	-0.1121
d2	-0.0918	1	0.1270	-0.0090	-0.0153	0.5848	-0.0488	0.9841	0.1706
d3	-0.0752	0.1270	1	0.9659	-0.0469	0.0095	-0.0138	0.0453	-0.0108
d4	0.1469	-0.0090	0.9659	1	0.1705	0.0971	0.0277	-0.1179	0.0171
d5	0.9941	-0.0153	-0.0469	0.1705	1	0.6366	-0.1006	-0.1254	-0.0266
d6	0.5515	0.5848	0.0095	0.0971	0.6366	1	0.3931	0.4535	0.5677
d7	-0.1698	-0.0488	-0.0138	0.0277	-0.1006	0.3931	1	-0.1450	0.9727
d8	-0.1901	0.9841	0.0453	-0.1179	-0.1254	0.4535	-0.1450	1	0.0652
d9	-0.1121	0.1706	-0.0108	0.0171	-0.0266	0.5677	0.9727	0.0652	1

A represent term vectors that show how terms are distributed across the entire collection, while the columns of A show which terms are present within a gene document. The optimal choice of k is application-dependent and is often empirically chosen. Typically, k is chosen so that $k \ll \min(m, n)$ [WCD03].

The goal of NMF is to approximate the original term by gene document space as accurately as possible with the factor matrices W and H. As noted earlier in Section 3.1, the SVD will produce the optimal low-rank approximation for any given rank s with respect to the Frobenius norm. Unfortunately, that optimality frequently comes at the cost of negative elements. The factor matrices of the NMF, however, are strictly nonnegative which may facilitate direct interpretability of the factorization. Thus, although an NMF approximation may not be optimal from a mathematical standpoint, it may be sufficient and yield better insight into the dataset than the SVD for certain applications.

Upon completion of NMF, the factor matrices W and H will, in theory, accurately approximate the original matrix A and yet contain some valuable information about the data

set in question. For example, Equation (3.2) can be rewritten as $a \approx Wh$. That is, columns a of A are approximated by linear combinations of the columns of W weighted by the columns h of H. Thus, W can be thought of as a basis of the data in A. Since only k columns are used to represent the original data and k is usually much smaller than n, W is a more accurate basis of A the closer that the columns of W are to true representations of the latent structure inherent in the data represented by A. That is, error in the overall approximation is minimized the closer that W becomes to representing the dominant features in the original data. As a result, W is commonly referred to as the *feature matrix* containing *feature vectors* that describe the themes inherent within the data, while H can be called a *coefficient matrix* since its columns describe how each document spans each feature and to what degree.

If NMF is applied to the sample term-document matrix in Table 2.2, one possible factorization is given in Tables 3.2 and 3.3; the approximation to the term-document matrix generated by mutliplying $W \times H$ is given in Table 3.4. The top-weighted terms for each feature are presented in Table 3.5. By inspection, the sample collection has features that represent *leukemia*, *alcoholism*, *anxiety*, and *autism*. If each document and term is assigned to its most dominant feature, then the original term-document matrix can be reorganized around those features. The restructured matrix typically resembles a block diagonal matrix and is given in Table 3.6.

Table 3.2: Feature matrix W for the sample collection.

	f1	f2	f3	f4
alcoholism	0.0006	0.3503		
anxiety			0.4454	
attack			0.4913	
autism		0.0030		0.8563
birth		0.1111	0.0651	0.2730
blood	0.0917	0.0538	0.3143	
bone	0.5220		0.0064	
cancer	0.1974	0.1906		
cells	0.1962		0.0188	
children		0.0019		0.5409
cirrhosis	0.0015	0.5328		
damage	0.2846			
defects		0.0662		0.4161
failure	0.0013	0.2988		
hypertension		0.1454	0.1106	
kidney	0.0013	0.2988		
leukemia	0.4513			
liver	0.0009	0.3366		
marrow	0.5220		0.0064	
pressure		0.066	0.6376	
scarring		0.208		
speech				0.4238
stress			0.6655	
tuberculosis	0.1962		0.0188	

Table 3.3: Coefficient matrix H for the sample collection.

	d1	d2	d3	d4	d5	d6	d7	d8	d9
f1		0.0409	1.6477	1.1382	0.0001	0.0007			
f2		1.3183			0.0049	0.6955	0.0003	0.9728	0.2219
f3	0.3836			0.0681	1.1933	0.3327			
f4						0.1532	0.9214		0.799

Table 3.4: Approximation to sample term-document matrix given in Table 2.2.

	d1	d2	d3	d4	d5	d6	d7	d8	d9
alcoholism		0.4618	0.0010	0.0007	0.0017	0.2436	0.0001	0.3408	0.0777
anxiety	0.1708			0.0303	0.5315	0.1482			
attack	0.1884			0.0334	0.5863	0.1635			
autism		0.0040				0.1333	0.7890	0.0029	0.6848
birth	0.0250	0.1464		0.0044	0.0783	0.1407	0.2516	0.1080	0.2428
blood	0.1206	0.0746	0.1511	0.1258	0.3754	0.1420		0.0523	0.0119
bone	0.0025	0.0214	0.8602	0.5946	0.0077	0.0025			
cancer		0.2593	0.3252	0.2247	0.001	0.1327	0.0001	0.1854	0.0423
cells	0.0072	0.0080	0.3233	0.2246	0.0224	0.0064			
children		0.0025				0.0842	0.4984	0.0019	0.4326
cirrhosis		0.7025	0.0024	0.0017	0.0026	0.3705	0.0002	0.5183	0.1183
damage		0.0116	0.4689	0.3239		0.0002			
defects		0.0873			0.0003	0.1098	0.3834	0.0644	0.3472
failure		0.3939	0.0022	0.0015	0.0015	0.2078	0.0001	0.2906	0.0663
hypertension	0.0424	0.1916		0.0075	0.1327	0.1379		0.1414	0.0323
kidney		0.3939	0.0022	0.0015	0.0015	0.2078	0.0001	0.2906	0.0663
leukemia		0.0185	0.7437	0.5137		0.0003			
liver		0.4437	0.0015	0.0011	0.0017	0.2341	0.0001	0.3274	0.0747
marrow	0.0025	0.0214	0.8602	0.5946	0.0077	0.0025			
pressure	0.2445	0.0870		0.0434	0.7612	0.2580		0.0642	0.0147
scarring		0.2742			0.0010	0.1446	0.0001	0.2023	0.0462
speech						0.0649	0.3905		0.3386
stress	0.2553			0.0453	0.7942	0.2214			
tuberculosis	0.0072	0.0080	0.3233	0.2246	0.0224	0.0064			

Table 3.5: Top 5 words for each feature from the sample collection.

f1	f2	f3	f4
bone	cirrhosis	stress	autism
marrow	alcoholism	pressure	children
leukemia	liver	attack	speech
damage	kidney	anxiety	defects
cancer	failure	blood	birth

Table 3.6: Rearranged term-document matrix for the sample collection.

	d3	d4	d2	d6	d8	d1	d5	d7	d9
bone	0.7520	0.7520							
cancer	0.4745		0.4745						
cells		0.6931							
damage	0.6931								
leukemia	1.0986								
marrow	0.7520	0.7520							
tuberculosis		0.6931							
alcoholism			0.4338	0.2737	0.2737				0.4338
cirrhosis			0.7520		0.7520				
failure			0.4745	0.4745					
hypertension				0.6931					
kidney			0.4745	0.4745					
liver			0.4745		0.4745				
scarring					0.6931				
anxiety						0.4745	0.4745		
attack							0.6931		
blood		0.3466		0.3466			0.3466		
pressure				0.4923			0.7804		
stress						0.4923	0.7804		
autism								0.7520	0.7520
birth				0.4745					0.4745
children								0.4745	0.4745
defects				0.3466				0.3466	0.3466
speech								0.6931	

3.2.1 Cost Function

Each iteration of NMF should improve its approximation of the original data matrix A with respect to some cost function. Equation (3.2) can be rewritten as

$$A = WH + C\left(A, WH\right),$$

where $C\left(A, WH\right)$ is a cost or error function. Typically, C is defined as either squared Euclidean distance or divergence. Squared Euclidean distance, or equivalently, the squared

Frobenius norm, is given by

$$\|A - WH\|_F^2 = \sum_{ij} \left(A_{ij} - (WH)_{ij}\right)^2.$$

This distance measure is zero if and only if $A = WH$ and is the primary measure used for derivations in this dissertation.

The divergence measure, given by

$$D\left(A\|WH\right) = \sum_{ij} \left(A_{ij} \log \frac{A_{ij}}{(WH)_{ij}} - A_{ij} + (WH)_{ij}\right),$$

is also zero if and only if $A = WH$ but is not symmetric in A and WH. If $\sum_{ij} A = \sum_{ij} WH = 1$, then the divergence measure is the Kullback-Leibler divergence or relative entropy [LS01].

NMF attempts to minimize the cost function C with respect to W and H subject to the constraints that both W and H must remain nonnegative. Both of the cost functions discussed are convex in either W or H, but not both variables together. As such, finding global minima to the problem is unrealistic—however, finding several local minima is within reason. Also, for each solution, the matrices W and H are not unique. This property is evident when examining $WDD^{-1}H$ for any nonnegative invertible matrix D [BBL+06].

3.2.2 Initialization

As previously noted, NMF may converge to a number of local solutions. Exactly to which solution NMF converges depends not only on the update rule employed, but on the starting point. The initial estimates for W and H affect both the final solution obtained as well as how fast NMF converges to that solution. Although many initialization strategies exist in the literature, in practice, most applications initialize both W and H to random positive entries.

Structured initialization is a way to speed convergence, however, the cost of performing the structured initialization must be weighed against the speedup in convergence as well as the solution quality. Since NMF attempts to uncover latent structure of the original data in a lower dimensional space, any method that can quickly approximate that structure will likely improve convergence over random initialization.

Other Methods

Centroid clustering based on Spherical K-Means was proposed in [WCD04] as a method to improve upon random initialization. Given k, Spherical K-Means can produce a centroid matrix M which can be arrived upon either by convergence or by completing a fixed number of iterations. With M, compute the nonnegative coefficient matrix N which minimizes $\|A - MN\|_F$ using the nonnegative least squares algorithm [LH74]. W and H are initialized to M and N, respectively. This method, however, suffers from a similar drawback to NMF—the initial partitioning of A is frequently assigned in some random fashion

[DFG01]. Like NMF, K-Means can be sensitive to the initial partitioning [Hei06], meaning that multiple runs to local optima will have to replace finding a global optima.[4]

Similar to the centroid clustering, [LMA06] proposes *SVD-centroid initialization* along with the next three schemes. While clustering on the original data matrix A can be relatively expensive, clustering on the SVD factor matrix V can be a faster alternative assuming the SVD is readily available. Once a clustering of V is achieved, for each centroid c_i of V, a new cluster centroid $\hat{c}_i$ is computed with the corresponding columns of the original data matrix A and entered into the initial column w_i of $W^{(0)}$. As a result, some of the structure inherent in the best-fit, low-rank space provided by the SVD and the nonnegativity of A can be exploited while the cost of directly clustering A is avoided. This method, however, still suffers from the same drawbacks as any K-Means-based algorithm.

Random Acol initialization also exploits the nonnegativity of A. Rather than forming a dense $W^{(0)}$ with positive entries, random Acol forms column w_i of $W^{(0)}$ by averaging p columns of A. As a result, much of the original sparsity in the data is preserved. Similarly, *random C initialization* (so named after its inspiration from the CUR decomposition [DKM06]) is another strategy that is basically random Acol applied to the longest columns of A rather than all of A. As a result, the most dense columns of A tend to be chosen as initial estimates for columns of $W^{(0)}$ since they will tend to have the largest 2-norm and will more likely be centroid centers.

Inspired by the term co-occurrence matrix presented in [San05], *co-occurrence initial-*

[4]Finding the optimal partitioning for any data set with respect to cluster coherence is known to be NP-complete [KPR98, GJW82].

ization forms the columns of $W^{(0)}$ as the "Topical Space" output from Algorithm 2 of that paper. Since the co-occurrence matrix AA^T is typically very large, initializing W with this method is impractical.

NNDSVD

[BG05] proposes the Non-negative Double Singular Value Decomposition (NNDSVD) scheme. NNDSVD aims to exploit the SVD as the optimal rank-k approximation of A. The heuristic overcomes the negative elements of the SVD by enforcing nonnegativity whenever encountered and is given in Algorithm 3.1.

[BG05] shows that the best nonnegative approximation to an arbitrary matrix M is the matrix M_+, which is formed by zeroing all the negative elements. Intuitively, for each SVD dimension $G^{(j)} = u_j v_j^T$, where $G^{(j)}$ denotes the rank-1 outer product of the jth left and right singular vectors, NNDSVD forms $G_+^{(j)}$, the best nonnegative approximation to that dimension. The most dominant singular triplet of that approximation is then calculated via the SVD and the corresponding singular vectors are inserted as a column or row in $W^{(0)}$ or $H^{(0)}$, respectively. The dominant singular triplet can be chosen to be nonnegative since the original matrix, G_+, is nonnegative.

The SVD can produce the optimal rank-k approximation of A as

$$A_k = U_k \hat{A}_k^T = \sum_{j=1}^{k} \sigma_j G^{(j)}$$

where $G^{(j)} = u_j a_j^T$ and $U_k = [u_1, \ldots, u_k], \hat{A}_k = [\sigma_1 a_1, \ldots, \sigma_k a_k]$. Simply seeding the

Algorithm 3.1 NNDSVD Initialization

Input: Term-by-Document Matrix A, Integer $k < \min(m, n)$
Output: Initial Factor Matrices $W^{(0)}$, $H^{(0)}$

Perform truncated SVD of A of k dimensions
Initialize first column of W with the first column of U
Initialize first row of H with the first row of V^T scaled by the first singular value
for $j = 2 : k$ **do**
  Form matrix $G^{(j)}$ by multiplying column j of U with row j of V^T
  Form matrix $G_+^{(j)}$ by setting negative elements of $G^{(j)}$ to 0
  Compute the maximum singular triplet (u, s, v) of $G_+^{(j)}$
  Set column j of W to u
  Set row j of H to v^T scaled by the jth singular value and s
end for

NMF by using the nonnegative elements of the singular vectors can be done, however, such a step can overconstrain the initialization as the product of the zeroed elements may itself be nonnegative (and hence store meaningful information). As a result, by computing the matrix $G^{(j)}$ corresponding to the jth singular vector pair and in turn forming rank-1 approximations to its nonnegative elements, more of the original data can be preserved. In any case, the big advantage of NNDSVD is that it exploits some of the structure inherent in the data and provides NMF with a static initialization—if NMF converges, it will converge to the same minima.

3.2.3 Update Rules

Once both W and H have been initialized, those initial estimates are iteratively improved upon, usually in either a multiplicative or an additive fashion.

Multiplicative Rules

The standard multiplicative update rule originally proposed by Lee and Seung and commonly referred to as the Multiplicative Method (MM) is given by

$$H_{cj} \leftarrow H_{cj}\frac{\left(W^TA\right)_{cj}}{\left(W^TWH\right)_{cj}}, \tag{3.3}$$

$$W_{ic} \leftarrow W_{ic}\frac{\left(AH^T\right)_{ic}}{\left(WHH^T\right)_{ic}}. \tag{3.4}$$

Since all the components of A, W, and H are guaranteed to be nonnegative, the updated matrices are guaranteed to be nonnegative. Lee and Seung also proved that the Euclidean distance cost function is nonincreasing when MM is applied and is invariant if and only if W and H are at a stationary point. The update factor becomes unity if and only if $A = WH$, which is as expected.

There are two practical extensions to MM that are commonly employed. To ensure numerical stability, a small positive number ϵ is added to the denominator of both Equations (3.3) and (3.4). Also, both H and W are updated "simultaneously" or by using the most up-to-date iterate rather than by updating each factor matrix independently of the other. As a result, faster convergence is usually observed.

Additive Rules

In addition to multiplicative rules, additive rules are also commonly used to update estimates of both factor matrices. Lee and Seung have shown that

$$H_{cj} \leftarrow H_{cj} + \eta_{cj} \left[\left(W^T A \right)_{cj} - \left(W^T W H \right)_{cj} \right]$$

is one such rule that can reduce the squared distance for some small values of η_{cj} (similar update for W). In fact, certain choices of η_{cj} can be shown to be equivalent to the multiplicative update. Many implementations of additive rules, however, ensure that nonnegativity is maintained by explicitly resetting negative values that result from the subtraction due to roundoff or other errors.

3.2.4 Stopping Criteria

Any of a number of methods can be employed to determine when to stop iterations. Commonly, NMF is run for a fixed number of predetermined iterations. Not surprisingly, this approach will usually cause NMF to over- or under-iterate. Another approach is to halt when the difference[5] between successive iterates is below a tolerance; that is, when $\|W_{old} - W_{new}\| < \tau$ and/or $\|H_{old} - H_{new}\| < \tau$. Obviously, the choice of tolerance τ determines the number of iterations completed. Similarly, some methods will calculate the objective function or the difference of objective function values between iterations. Al-

[5] Assume Frobenius matrix norm unless otherwise specified.

though with any method that requires threshold checking of differences between iterates, those iterates in question must be stored, and frequently extra computation must take place.

3.2.5 Additional Constraints

NMF with the MM update proposed by Lee and Seung is guaranteed to converge to a local solution, however, additional constraints can be placed on the cost function to help ensure any of a variety of behaviors such as faster convergence or certain structural properties of the factor matrices. Unfortunately, once additional constraints are placed on the cost function, most update rules are no longer guaranteed to converge. In general, additional constraints redefine the overall minimization problem to a variant of

$$\min_{W,H} \|A - WH\|_F^2 + \alpha J_1(W) + \beta J_2(H), \tag{3.5}$$

where α and β are parameters and $J_1(W)$ and $J_2(H)$ are functions that describe the additional constraints placed on W and H.

The framework presented in Equation (3.5) is not all-inclusive—there can be other constraints that may not fit this form. For example, many algorithms require that columns of either W or H be normalized, either after each iteration or after convergence has been reached. Normalization will, in theory, force unique solutions at each minima. Unfortunately, however, this normalization usually comes at the cost of convergence speed, i.e. number of iterations performed.

Two additional constraints that do follow the framework presented in Equation (3.5)

are the notions of *smoothing* and *sparsity*. In each case, the constraints are intended to guarantee structural properties in the factor matrices, however, for many parameter settings a problem can quickly become overconstrained and may not converge. If the solution does converge, however, the resulting factor matrices can often lead to more intuitive interpretations of the data.

Smoothing

Smoothing a solution is meant as a filter to reduce *noise*. In the case of NMF, smoothing a factor matrix reduces the number and relative magnitude of high amplitude components. In effect, the solution is more likely to have all of its components with similar magnitude. To accomplish this in W, set

$$J_1(W) = \|W\|_F^2 .$$

This can also be done to H, depending on the results desired within the problem context. [PPPG04] showed that if smoothing is enforced on both W and H, the update rule becomes

$$\begin{aligned} H_{cj} &\leftarrow H_{cj} \frac{\left(W^T A\right)_{cj} - \beta H_{cj}}{\left(W^T W H\right)_{cj} + \epsilon}, \\ W_{ic} &\leftarrow W_{ic} \frac{\left(A H^T\right)_{ic} - \alpha W_{ic}}{\left(W H H^T\right)_{ic} + \epsilon}, \end{aligned}$$

where $\alpha, \beta \in \Re$ are parameters that denote the degree to which smoothness will be enforced.

Sparsity

Whereas smoothing brings the relative magnitudes of a solution closer together, enforcing *sparsity* constraints aims to accomplish nearly the opposite effect. Based on nonnegative sparse coding, sparsity constraints within NMF was originally proposed in [Hoy02]. Unfortunately, sparseness was controlled implicitly, and parameter settings had to be determined empirically to achieve the desired level of sparseness. Later in [Hoy04], Hoyer improved upon his original idea by explicitly adjusting sparseness, defined by

$$\text{sparseness}\,(x) = \frac{\sqrt{n} - \left(\sum |x_i|\right) / \sqrt{\sum x_i^2}}{\sqrt{n} - 1}, \tag{3.6}$$

where n is the length of vector x. This measure, which is based on the L_1 and L_2 norms, evaluates to one if and only if exactly one nonzero component exists in x. On the other hand, a value of zero can only be attained if all components have identical magnitude.

Hoyer's NMF algorithm with sparseness constraints is a projected gradient descent algorithm that utilizes both multiplicative and additive update rules in conjunction with a projection operator to produce an approximation. [PPA07] uses the sparseness defined in Equation (3.6) to form a different approach using only multiplicative updates via the following derivation. Assuming that $\bar{H}$ denotes the vector formed by stacking the columns of H, the general objective function in Equation (3.5) can be rewritten to incorporate sparseness as

$$F(W,H) = \frac{1}{2}\left\|A - WH\right\|_F^2 + \frac{\beta}{2}\left(\omega_H \left\|\bar{H}\right\|_2 - \left\|\bar{H}\right\|_1\right)^2,$$

where ω_H is derived from Equation (3.6) and is computed as

$$\omega_H = \sqrt{kn} - \left(\sqrt{kn} - 1\right) \text{sparseness}\left(H\right).$$

The partial derivatives with respect to H_{ij} are

$$\begin{aligned}
\frac{\partial}{\partial H_{ij}}\left(\|A - WH\|_F^2\right) &= -\left(W^T A\right)_{ij} + \left(W^T W H\right)_{ij} \\
\frac{\partial}{\partial H_{ij}}\left(\left(\omega_H \left\|\bar{H}\right\|_2 - \left\|\bar{H}\right\|_1\right)^2\right) &= 2\omega_H^2 H_{ij} - 2\omega_H \left(\frac{\left\|\bar{H}\right\|_1}{2\left\|\bar{H}\right\|_2} H_{ij} + \left\|\bar{H}\right\|_2\right) + 2\left\|\bar{H}\right\|_1
\end{aligned}$$

and can be simplified to

$$\frac{\partial}{\partial H_{ij}}\left(F\left(W, H\right)\right) = -\left(W^T A\right)_{ij} + \left(W^T W H\right)_{ij} + \beta\left(c_1 H_{ij} + c_2\right),$$

where

$$c_1 = \omega_H^2 - \omega_H \frac{\left\|\bar{H}\right\|_1}{2\left\|\bar{H}\right\|_2} \tag{3.7}$$

$$c_2 = \left\|\bar{H}\right\|_1 - \omega_H \left\|\bar{H}\right\|_2. \tag{3.8}$$

Equivalently, this is expressed in matrix form as

$$\nabla F\left(W, H\right) = -W^T A + W^T W H + \beta\left(c_1 H + c_2 E\right),$$

where E denotes a $k \times n$ matrix of ones. Following [LS01], consider

$$H_{ij} = H_{ij} - \phi_{ij} \frac{\partial}{\partial H_{ij}} \left(F\left(W, H\right) \right)$$

as the formula for updating H, where

$$\phi_{ij} = \frac{H_{ij}}{\left(W^T W H\right)_{ij}}.$$

By substitution, the update rule then becomes

$$H_{ij} = H_{i}j \left(\frac{\left(W^T A\right)_{ij} - \beta \left(c_1 H_{ij} + c_2 E_{ij}\right)}{\left(W^T W H\right)_{ij}} \right),$$

where c_1 and c_2 are defined in Equations (3.7) and (3.8), respectively. Following a similar procedure, sparsenesss constraints can be placed upon W. The update rule derived then becomes

$$W_{ij} = W_{ij} \left(\frac{\left(AH^T\right)_{ij} - \alpha \left(c_1 W_{ij} + c_2 E_{ij}\right)}{\left(WHH^T\right)_{ij}} \right),$$

where

$$\omega_W = \sqrt{mk} - \left(\sqrt{mk} - 1\right) \text{sparseness}\left(W\right)$$

replaces ω_H and W replaces H in the definition of c_1 and c_2.

3.3 Comparison Between LSI and NMF

Both LSI and NMF are dimension reduction techniques that aim to compress the original space to gain insight or reveal some underlying structure in the data. The SVD in LSI produces the mathematically optimal low-rank approximation for any k; that approximation is unique, and the basis produced is orthogonal. Given the orthogonal bases produced, data can be projected into the low-rank space, and queries can be processed. That orthogonality, however, comes at the price of negativity and density. The basis vectors will be dense (hence not accurately reflecting the original data) and will have negative components meaning that the axes of the basis vectors cannot be easily identifiable in terms of the original data.

Since the NMF factor matrices usually maintain a good deal of sparsity to reflect the original data and are guaranteed to remain nonnegative, NMF, on the other hand, produces "basis" vectors that are easily identifiable in the original space. This sparsity can lead to a reduction in storage requirements [LMA06]. Unfortunately, NMF has convergence problems when additional constraints are enforced. Also, it is sensitive to its initialization—for example, if the initialization sets an element to zero, then, given the mulitiplicative update rule, that element will remain zero throughout all the subsequent iterations. This property may have a negative effect on the overall solution quality. Speaking of solution quality, NMF is not guaranteed to find a global minima. For any given minima, an infinite number of solutions exist, and NMF will not necessarily produce the same solution on any two given runs. As an additional burden, NMF and its variants have a large number of param-

eters that must be set, and those parameters may not have an easily explainable result or may not correlate to similar parameter settings. For example, unlike the SVD, there is no straightforward way to relate feature vector k with $k + 1$.

While both techniques have their shortcomings, their strengths can complement each other nicely. While there is no straightforward way to transform one factorization into another, when used in tandom they can uncover different properties of the data. The density of the SVD lends itself to global support—vectors in that space will produce similarities to others that would not have been obvious in the original data. As a result, LSI seems to be the optimal choice when viewing the entire data set as a whole or when trying to uncover latent structure. NMF, on the other hand, excels at clustering and classification and showing why a relationship between two entities exist. NMF is known for its *parts-based* factorization, meaning that it shows local rather than global properties of the data. Rather than uncovering an optimal space that makes no sense in terms of the original data, NMF can identify where data groups exist and suggest labels for that group.

Chapter 4

Performance Evaluation

Both IR (Chapter 2) and dimension reduction techniques (Chapter 3) have been introduced—this chapter will show how those techniques can be interpreted and automatically evaluated in biological applications. LSI, discussed in Section 3.1, can be used to create a global *picture* of the data automatically. In this particular context, that global picture is most meaningful when viewed as a hierarchical tree. Once a tree is built, a labeling algorithm can be applied to identify branches of the tree. Finally, a "gold standard" tree and a standard performance measure that evaluates the quality of tree labels must be defined and applied.

4.1 Hierarchical Tree Construction

One popular tool for visualizing the evolution of taxa over time is an *evolutionary tree*, also known as a phylogeny, phylogenetic tree, or hierarchical tree. Formally, a tree is a cycle-free graph $G = (V, E)$ where V is the set of vertices (or nodes) and E is the set of

edges. That is, any two nodes in the graph have exactly one unique path between them. A simple phylogeny is depicted in Figure 4.1.

If two vertices are connected by an edge, then each of the vertices is said to have that edge *incident upon* it; the *degree* of any vertex is the number of edges incident upon it. Each node of degree 1 (which are denoted by lettered nodes in Figure 4.1) are called *leaf nodes*; every other node in a phylogeny is commonly referred to as an interior or ancestral node. Since information is only known about the taxa a priori, the phylogeny must be *inferred*. If a phylogeny truly depicts ancestral relationships, then the top node becomes the *root node* of a *rooted tree*, and the tree usually takes on the form of a directed acyclic graph (DAG) with a direction pointing away from the root.

Generally speaking, taxa can refer to any of a variety of entities such as organisms, genes, proteins, and languages, while the edges can represent parent–child relationships along with additional information. Exactly what information edges represent is usually determined by the input data and the type of tree building method employed. Regardless

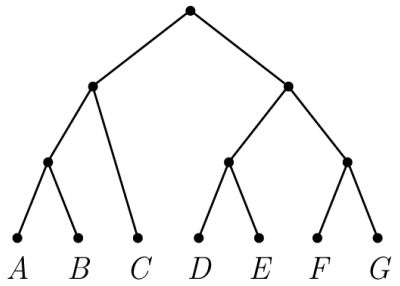


Figure 4.1: A sample phylogeny.

of the type of method employed, most phylogeny inference schemes adhere to the principle of minimum evolution [KSZ71]. Namely, they follow Occam's razor and prefer the simplest solution over more complicated ones. Although many different tree building algorithms exist, most can usually be classified as sequence- or distance-based methods, and a representative few are discussed in this chapter.

4.1.1 Sequence-Based Methods

As the name suggests, sequence-based methods assume that the input taxa are a set of sequences, commonly of DNA or amino acids. Most sequence-based methods operate under the assumptions that any two characters are mutually independent within a given sequence, and that evolution occurs independently after two sequences diverge in a tree [Sha01]. Under these assumptions, a tree can be built where the leaves are the sequences and the edges describe some evolutionary phenomenon such as gene duplication, mutation, or speciation. For example, the binary input given in Table 4.1 can be represented by the tree in Figure 4.2 [Fel82], where each labeled edge denotes the character state at which there is a change. Using the same convention, this example can easily be extended to DNA, protein, and other sequence data.

One easy evaluation for a phylogenetic tree is to compute the number of character changes depicted by the tree, also known as the *Hamming* or *edit distance*. In Figure 4.2, the tree has 10 such changes (computed by adding the number of changes denoted along the edges). Methods that attempt to minimize this score are called *parsimony* or minimum-

Table 4.1: Sample data set of binary strings.

	Character					
Taxon	1	2	3	4	5	6
A	1	1	0	0	0	0
B	0	0	1	0	1	1
C	1	1	0	0	1	1
D	0	0	1	1	0	0
E	0	1	0	0	1	1

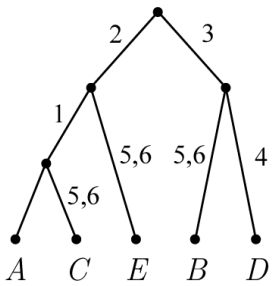


Figure 4.2: A parsimony tree based on the data in Table 4.1.

evolution methods.

The problem of finding a most parsimonious labeling of a tree given a topology is known as the *Small Parsimony Problem*. Fitch presented an algorithm in [Fit71] to solve the problem, while Sankoff presented a similar solution to the weighted problem using a dynamic programming approach [San75] that can solve the problem in $O(mnk)$, where m is the sequence length, n is the number of taxa, and k is the number of values any character state can assume.

The *Large Parsimony Problem*, on the other hand, is $\mathcal{NP}$-complete. That is, given a set of n sequences of length m, the problem of finding both the tree and the labeling that produces the most parsimonious score possible has an exponentially large solution space. One common search heuristic employed to find local solutions is called *nearest neighbor interchange* (NNI) [Rob71]. One NNI can be employed by selecting an edge and hypothetically removing the two nodes it connects. The resulting subtrees (four will occur if the an internal edge is removed and the tree is binary) are then swapped in the original tree and the corresponding parsimony computed. The interchange with the lowest parsimony score is enacted, and the process repeated for all edges in the tree. A sample of three possible neighbors is given in Figure 4.3, where the dashed edge represents the edge defining the subtrees of the NNI [JP04].

Not far removed from parsimony is the notion of *compatibility*. Rather than including all taxa and trying to minimize the parsimony score, compatibility methods attempt to maximize the number of characters that can be used to create a perfect phylogeny. A

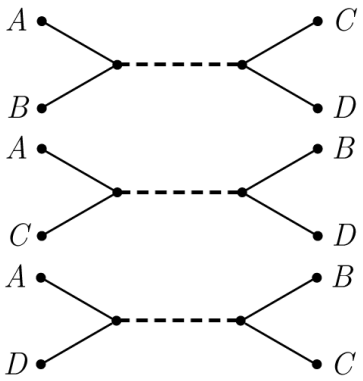


Figure 4.3: Three neighbor trees induced by a NNI about the dashed edge. The labeled nodes can represent full subtrees.

perfect phylogeny exists if and only if an edge representing a particular state change forms a subtree; the tree given in Figure 4.2 is not a perfect phylogeny because multiple edges exist that denote changes in states 5 and 6, and any one of those edges does not induce a subtree. The *Large Compatibility Problem*, which is similar to its parsimony counterpart, attempts to find the maximum number of characters that will induce a perfect phylogeny as well as the phylogeny it induces. This problem, predictably, is also known to be $\mathcal{NP}$-complete in the when a variable number of states is allowed [BFW92].

4.1.2 Distance-Based Methods

With sequence-based methods, the input data is assumed to be a set of character strings, and the edges of the inferred tree models the divergences in those strings. On the other hand, distance-based methods assume that the input is a matrix of pairwise distances between

n taxa, and the edges of the phylogeny produced represent distances between nodes. As with sequence-based methods, the independent evolution after speciation is assumed. An example tree built upon distance data given in Table 4.2 is given in Figure 4.4.

Typically in the biological context, the distance data generated is the edit distance between two sequences or some function thereof. Similar to the parsimony problem, the small version of the distance problem (determining branch lengths given a tree) can be solved relatively easily, but the more practical large version (determining the optimal tree along with branch lengths) is $\mathcal{NP}$-complete. As can be expected, a variety of heuristics exist, with many of them being least squares approaches. That is, the objective function

$$LSQ(T) = \sum_{i=1}^{n} \sum_{j \neq i} w_{ij} \left(D_{ij} - d_{ij}\right)^2 \tag{4.1}$$

is to be minimized, where D_{ij} is the observed distance, d_{ij} is the predicted distance in tree T, and w_{ij} is some weight, usually set to 1. Fitch in [FM67] proposed a bottom-up method that finds a local optimum to this function in polynomial time, however, since that algorithm operates in $O(n^4)$, clustering more than a handful of taxa becomes an impractical

Table 4.2: Sample pairwise distance data for the tree given in Figure 4.4 [Sha01].

	A	B	C
A	0	0.08	0.45
B	0.08	0	0.43
C	0.45	0.43	0

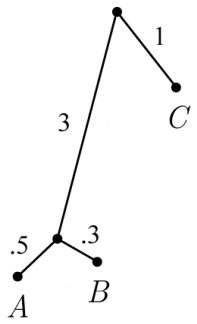


Figure 4.4: A tree where the branch lengths correspond to the distances given in Table 4.2 [Sha01].

exercise.

Saitou and Nei proposed the popular neighbor-joining (NJ) method in [SN87] as a heuristic that infers phylogenies from a pairwise distance matrix in $O(n^3)$ time. NJ begins by assigning each taxon to its own cluster (commonly referred to as a "star" configuration). The two closest taxa are joined into one cluster with branch lengths computed by the method described in [FM67], and the new $(n-1) \times (n-1)$ distance matrix is recomputed with the new cluster substituted for the two joined clusters. This process is repeated until all taxa are in one cluster. NJ has since been refined with algorithms such as BIONJ in [Gas97] and WEIGHBOR in [BSH00], but the time complexity of all of NJ's variants remains $O(n^3)$.

By applying a greedy approach to generate an initial tree in $O(n^2)$ time and applying NNIs, Desper and Gascuel in [DG02] were able to generate accurate phylogenies in $O(n^2)$ time. The Greedy Minimum Evolution (GME) algorithm is used to initialize the tree,

whereby a tree of size 3 is initialized and each node is added by replacing the edge that gives the resulting tree the minimum length. This computation can be accomplished in $O(n^2)$ time, which is the input size. The FastNNI algorithm then improves upon the initial tree by traversing through all the internal edges and performing an NNI if it results in a better tree. FastNNI runs in $O(n^2 + pn)$, where p is the number of swaps needed. Since p is usually much smaller than n, this can be done in $O(n^2)$ time.

BME and BNNI, balanced versions of GME and FastNNI that give sibling subtrees rather than taxa equal weight, were also implemented and were found to have running times of $O(n^2 \times \text{diam}(T))$ and $O(n^2 + np\ \text{diam}(T))$, respectively, where $\text{diam}(T)$ is the diameter (number of edges) of the tree T. In the worst case, $\text{diam}(T)$ can evaluate to n, however, in practice it typically is equivalent to $\log(n)$ or $\sqrt{n}$. Either GME or BME coupled with FastNNI or BNNI were found to have trees that were close to or better than the topological accuracy provided by NJ, and in all cases were produced faster.

4.2 Labeling Algorithm

Once a hierarchy is produced, labeling the internal nodes can help describe the nature of the relationships represented by the hierarchy structure; placing labels on the internal nodes can be accomplished by any number of methods, although relatively few well-established automated methods exist. In the field of machine learning, the problem where each taxa can belong to any number of classes is known as multi-label hierarchical classification. One such labeling algorithm from this field uses a Bayesian framework based upon support

vector machine classifiers to make functional predictions [BST06]. In the particular case of SGO, since the SVD is readily available, a concatenation of all the documents that are members of each subtree that each node induces can be formed and the near terms in the LSI-based vector space found via Equation (3.1). The resulting labeling may not initially satisfy many of the consistency properties desired in the machine learning field, but it provides a basis against which to compare NMF labels.

To apply labels from the NMF, a bottom-up approach is used. For any given document, the highest weighted row index of H is chosen, which corresponds to the most dominant feature vector in W for that document. The terms in that feature vector are assigned to that document and weighted by the corresponding coefficient in H. As a result, each document has a ranked list of m terms with weights associated with it. In practice, this list is thresholded to a fixed number of terms.

Once each document has its associated terms, they can be assigned to the hierarchy by simply inheriting them upward. That is, when two siblings A and B have parent C, each term from both A and B are added to the list associated with C. As a result, terms common to both A and B will be more likely to be highly weighted within C. This process is iteratively applied to the entire tree. At higher levels within the tree, this method tends to give more weight to the denser subtrees and to the more commonly used words. Consequently, broader terms tend to be dominant closer to the root, which is intuitive. The full algorithm is presented in Algorithm 4.1.

This bottom-up approach can be modified to incorporate more than just the dominant

Algorithm 4.1 Bottom–up labeling algorithm.

Input: Factor Matrices $W_{m\times k}$ and $H_{k\times n}$
Hierarchical Tree T
p, threshold on number of terms to inherit
Output: Labeled Tree T'

$nodes$ = leaf nodes of T
for $i = 1 : n$ **do**
Determine $j = \arg\max_{j<k} H_{ji}$
Assign top p terms from jth column of W to node i
i.e., for term index q, assign weight $W_{qi} * H_{ji}$
end for
while $|nodes| > 1$ **do**
Remove two sibling nodes A and B (with parent C) from $nodes$
Merge term lists from A and B (add weights for common terms)
Assign terms to C
Push C onto $nodes$
end while

feature vector—as many as k feature vectors can be used to associate terms with each document. Doing so would make the terms a closer approximation to the original term-document matrix and would have the end effect of assigning more topics to each document. Making this modification can produce more accurate labels depending upon the exclusivity of the assignments within the dataset. Also, averaging rather than simply adding term weights places equal importance to the terms associated with each subtree. Another variation may also scale the inherited terms by the corresponding branch lengths of the tree. The end result would give higher preference to terms that are associated with genes that are closer to each other, although some heurisitic would have to be applied to accommodate negative branch lengths.

This algorithm, which is in many respects analogous to NJ, can be slightly modified

and applied to any tree where a ranked list can be assigned to each taxon. For example, by querying the SVD-generated vector space for each document, a ranked list of terms can be created for each document and the tree labeled accordingly. The "correct" MeSH labeling is generated in a similar manner. Using the subtree in Figure 4.5 and the log-entropy weights from the *50TG_MeSH* meta-collection to generate a ranked list, MeSH headings are inherited up the tree by averaging the weights of each sibling node to give each subtree an equal contribution to its parent.[1] The top 5 MeSH headings at each stage are depicted in Figure 4.6.[2] By using Algorithm 4.1 or a derivative of it and assuming the initial ranking procedure is accurate, any ontology annotation can be enhanced with terms from the text it represents.

4.3 Recall Measure

Once labelings are produced for a given hierarchical tree, a measure of "goodness" must be calculated to determine which labeling is the "best." When dealing with simple return lists of documents that can classified as either relevant or not relevant to a user's needs, IR methods typically default to using precision and recall to describe the performance of a given retrieval system. *Precision* is defined as

$$P = \frac{|R|}{|L|},$$

[1] The *50TG_MeSH* collection contained 9,126 MeSH headings.
[2] For evaluation purposes, only the top-ranked MeSH heading is used to label the "correct" tree.

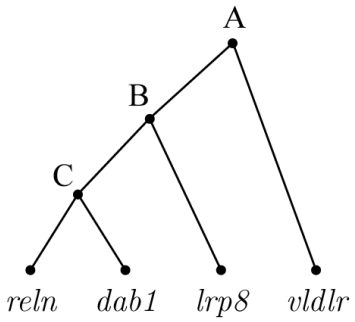


Figure 4.5: *Reln* subtree extracted from the *50TG* tree.

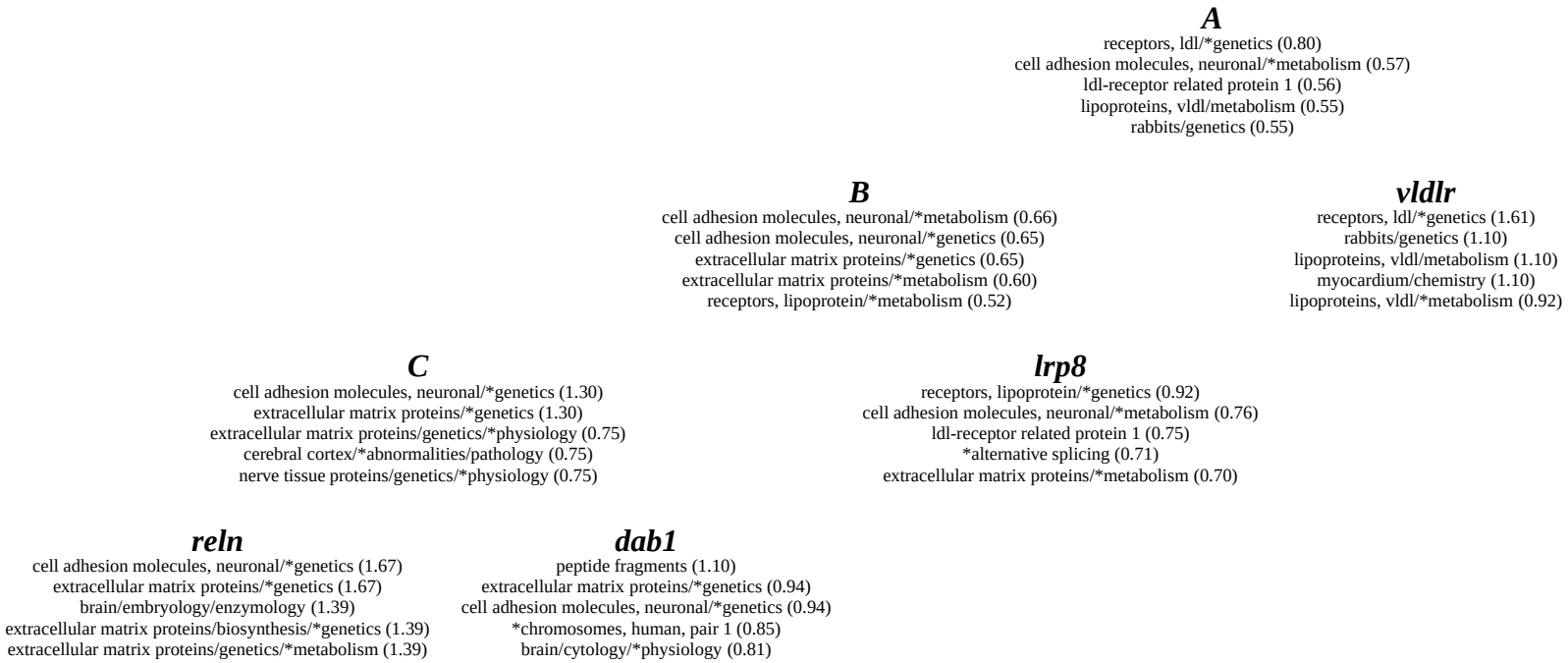
where R denotes the set of relevant documents within the return list and L is the set of returned documents. *Recall*, on the other hand, is denoted by

$$R = \frac{|R|}{|T|},$$

where T denotes the set of all relevant documents [BB99]. In other words, precision measures how accurately a system returns relevant documents, while recall quantifies the system's coverage of all relevant documents. The goal of all information retrieval systems is to have high levels of precision at high levels of recall. Unfortunately, however, as the level of recall rises, precision tends to fall.

To depict a system's performance, precision-recall graphs are constructed whereby precision is plotted against the standard decile ranges (i.e. 10%, 20%, etc) of recall. To condense this into a single number, the concept of the n-point interpolated *average precision* is introduced as

Figure 4.6: Top 5 MeSH labels associated with each node of the tree in Figure 4.5. The number in parentheses indicate the weight associated with each MeSH heading; the weights assigned to each leaf node are the log-entropy weights from the *50TG_MeSH* collection.



$$AP = \frac{1}{n} \sum_{i=0}^{n-1} \tilde{P}\left(\frac{i}{n-1}\right),$$

where $\tilde{P}(x)$ is called the pseudo-precision and is the maximum precision up to the ith document, and n is typically chosen to be eleven.

Another measure of a system's performance can be given by harmonic mean of precision and recall and is known as the F-measure. This measure, given by

$$F = \frac{2(P \times R)}{(P + R)},$$

is a nonnegative fraction that achieves unity if and only if both precision and recall are one [BYRN99]. A weighted version of F can also be applied to stress either component over the other.

Unfortunately in this context, however, the set being measured is not as a simple as a return list. In the case of SGO, one labelled hierarchy must be compared to another. Surprisingly, relatively little work has been done that addresses this problem. Graph similarity algorithms exist, but most all of those algorithms compare differing topologies—the labels on the nodes of the graphs are artifacts that are usually ignored. When comparing labels that occur in a hierarchical fashion, a single number evaluation is wanted to allow comparison of large amounts of data, but the ability to measure different properties of the labeling at some point would also be useful. In this context, identifying how well a labeling performed

at different levels within the hierarchy could help evaluate the system more accurately.

Kiritchenko in [Kir05] proposed the hierarchical precision and recall measures, denoted as hP and hR, respectively. If C denotes the set of classes into which nodes are classified (in SGO's case, C refers to text words), then for each node i,

$$\begin{aligned} hP &= \frac{\sum_i \left|\hat{C}_i \cap \hat{C}'_i\right|}{\sum_i \left|\hat{C}'_i\right|}, \\ hR &= \frac{\sum_i \left|\hat{C}_i \cap \hat{C}'_i\right|}{\sum_i \left|\hat{C}_i\right|}, \end{aligned}$$

where $C_i \subseteq C$ is the correct classification, $C'_i \subseteq C$ is the assigned classification, and the $\hat{\cdot}$ operation denotes the inclusive union of all labels in the ancestor set. This measure assumes that a classfication hierarchy exhibits *hierarchical consistency*; a hierarchy is deemed consistent if each label includes complete ancestor sets. Formally, for each label $c_k \in C_i$ for node C_i, $c_j \in Ancestors(c_k)$ implies $c_j \in C_i$ [KMF05]. In other words, hierarchical consistency takes the parent–child relationship exhibited by most hierarchical trees to the extreme by requiring that all node labels must be inherited down to all their respective children nodes.

While hierarchical consistency is an intuitive property of many trees, many hierarchies in practice do not explicitly enforce this constraint. If this constraint were required, leaf nodes could potentially have very large label sets while the ancestor nodes would have fewer descriptors. Of course, this can be avoided by adhering to a convention that places each label at its most ancestral node. Although hierarchical consistency is intuitive as leaf

nodes often represent the most restrictive nodes within the tree, if applied to a true tree where taxa may belong to more than one parent, the labeling may produce unintended results. For example, consider the tree given in Figure 4.7. If node C could be feasibly considered to be a descendant of both node X and node Y (i.e., C contains labels that are present in both X and Y) where the labels of X and Y are disjoint, then the "tree" (more precisely, the graph) given in Figure 4.8 could more accurately describe the evolutionary relationship. If the root node can be ignored for the purposes of this argument, then enforcing hierarchical consistency on the tree would force Y to contain labels of X and not vice versa. In the case of the graph, however, hierarchical consistency can be applied easily. This implies that the notion of hierarchical consistency is robust within hierarchical graphs, but when mutually independent evolution is assumed after each node, then this constraint may produce unintended results. Related to this point, the unweighted hP and hR measures penalize misclassifying higher nodes more than lower ones. As such, these measures can be considered top-down approaches.

Also, while these measures condense the information in a tree into a single number,

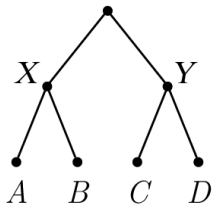


Figure 4.7: Simple tree that is hierarchically consistent if all labels in X occur in A and B and all labels in Y occur in C and D.

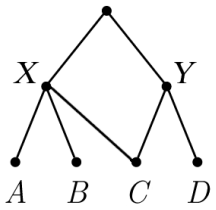


Figure 4.8: Hierarchical graph extended from Figure 4.7.

they do not show how well a labeling performs as a function of tree depth. In the case of SGO, this information is crucial as varying parameter settings within the NMF are expected to affect performance at different depths within the tree. To address this issue, a simple measure for gauging accuracy is to find the recall at each node. Since, in practice, words in a label are unranked, the notion of precision carries little meaning. The recall values at each level within a tree can be averaged to show how accurately a labeling is a function of tree depth, and those can in turn be averaged to produce a single value that captures how well the labeling performed overall.

4.4 Feature Vector Replacement

When working with gene documents, many cases exist where the terminology used in MeSH is not found within the gene documents themselves. Even though a healthy percentage of the exact MeSH terms may exist in the corpus, the term-document matrix is so heavily overdetermined (i.e., the number of terms is significantly larger than the number of documents) that expecting significant recall values at any level within the tree becomes

unreasonable. This is not to imply that the terms produced by NMF are without value. On the contrary, the value in those terms is exactly that they may reveal what was previously unknown. For the purposes of validation, however, some method must be developed that enables a user to discriminate between labelings even though both have little or no recall with the MeSH-labeled hierarchy. In effect, the vocabulary used to label the tree must be controlled for the purposes of validation and evaluation.

To produce a labeling that is mapped into the MeSH vocabulary, the top r globally-weighted MeSH headings are chosen for each document; these MeSH headings can be extracted from the MeSH meta-collection (see Section 2.5). By inspection of H, the dominant feature associated with each document is chosen and assigned to that document. The corresponding top r MeSH headings are then themselves parsed into tokens and assigned to a new MeSH feature vector appropriately scaled by the corresponding coefficient in H. The feature vector replacement algorithm is given in Algorithm 4.2.[3]

Once full MeSH feature vectors have been constructed, the tree can be labeled via the procedure outlined in Algorithm 4.1. As a result of this replacement, better recall can be expected, and the specific word usage properties inherent in the MeSH (or any other) ontology can be exploited.

[3]Note that m' is distinguished from m since the dictionary of MeSH headings will likely differ in size and composition from the original corpus dictionary. The number of documents, however, remains constant.

Algorithm 4.2 Feature Vector Replacement Algorithm

Input: MeSH Term-by-Document Matrix $A'_{m' \times n}$
Factor Matrices $W_{m \times k}$ and $H_{k \times n}$ of original Term-by-Document Matrix $A_{m \times n}$
Global weight vector g',
Threshold r number of MeSH headings to represent each document
Output: MeSH feature matrix W'

for $i = 1 : n$ **do**
  Choose r top globally–weighted MeSH headings from ith column of A'
  Determine $j = \arg \max_{j<k} H_{ji}$
  for $h = 1 : r$ **do**
    Parse MeSH heading h into tokens
    Add each token t with index p to w'_j, the jth column of W'
      i.e., $W'_{pj} = W'_{pj} + g'_p \times H_{ji}$
  end for
end for

Chapter 5

Results

The evaluation of the factorization produced by NMF is non-trivial as there is no set standard for examing the quality of basis vectors produced. For example, [CCSS$^+$06] performed several NMF runs then independently asked domain experts to interpret the resulting feature vectors. This chapter attempts to evaluate NMF by two distinct, automated methods. First, the mathematical properties of the NMF runs are examined, then the accuracy of the application of NMF to hierarchical trees is scrutinized for each collection. In both cases, the effects of each parameter is discussed.

5.1 Data Sets

Five data sets were used to test the effectiveness of NMF. Each group of genes specified were related to a specific biological function, with some extraneous or "noisy" genes unrelated to that function possibly added. The full list of genes in each dataset can be found in

Appendix A. Each token in every document collection was required to be present at least twice in a single document and at least twice throughout the collection.

For each collection, two different initialization strategies were used: the NNDSVD (Section 3.2.2) and randomization. Five different random trials were conducted while three were performed using the NNDSVD method. Although the NNDSVD produces a static starting matrix, different methods can be applied to remove zeros from the initial approximation to prevent them from getting "locked" throughout the update process. Initalizations that maintained the original zero elements are denoted NNDSVDz, while NNDSVDa, NNDSVDe, and NNDSVDme substitute the average of all elements of A, ϵ, or $\epsilon_{machine}$, respectively, for those zero elements; ϵ was set to 10^{-9} and was significantly smaller than the smallest observed value in either H or W (typically around 10^{-3}), while $\epsilon_{machine}$ was the machine epsilon (the smallest positive value the computer could represent) at approximately 10^{-324}. Both NNDSVDz and NNDSVDa are mentioned in [BG05], however, NNDSVDe and NNDSVDme were added as natural extensions to NNDSVDz that would not suffer from the restrictions of locking zeros due to the multiplicative update.

Four different constraints were placed on the iterations in addition to the basic NMF: smoothing on W, sparsity on W, smoothing on H, and sparsity on H. In the cases where extra constraints were added, the parameter values tested were 0.1, 0.01, and 0.001; sparsity values of 0.1, 0.25, 0.5, 0.75, and 0.9 were tested where applicable. The number of feature vectors, k, was the remaining free parameter and was assigned the values of 2, 4, 6, 8, 10, 15, 20, 25, and 30. When all parameter settings are considered, NMF was applied to

each collection over 2,500 times. The only exception is the 50TG collection, where over 7,500 runs were performed to inspect the effect of the various column normalizations on the factorizations.[1] For each value of k, a total of 135 runs were performed when sparsity was enforced on either either W or H, and 27 runs were performed when smoothness was imposed on either W or H.

Each of the more than 30,000 NMF runs iterated until it reached 1,000 iterations or a stationary point in both W and H. That is, at iteration i, when $\|W_{i-1} - W_i\|_F < \tau$ and $\|H_{i-1} - H_i\|_F < \tau$, convergence is assumed. The parameter τ was set to 0.01. Since convergence is not guaranteed under all constraints, if the objective function increased betweeen iterations, the factorization was stopped and assumed not to converge.

The log-entropy weighting scheme discussed in Chapter 2 was applied to generate token weights for each collection. To simulate the effect of more discriminating token selection algorithms and to reduce the relative size of the dictionary, all tokens whose global weight fell below the threshold of 0.8 were added to a stoplist, and the collection was reparsed and reweighted. Any collection where this threshold was applied is denoted with a ".8" appended to it (e.g., *50TG* refers to the original collection, while *50TG.8* denotes the reweighted *50TG* collection after all tokens with an initial global weight of less than 0.8 were removed). The term counts concerning each collection before and after thresholding are presented in Table 5.1.

The *50TG* collection is comprised of 50 genes that are associated with the $Reelin$ sig-

[1] Only two random runs were performed when any of the columns were normalized.

Table 5.1: Corpus size for the five collections considered. The numbers in parentheses show the term count after removing terms with a global weight of less than 0.8.

Collection	# Terms		# Documents
50TG	8,750	(3,073)	50
115IFN	8,758	(3,250)	115
Math1	3,896	(1,414)	46
Mea	3,096	(1,204)	45
Sey	5,535	(1,997)	35

naling pathway, and evidence suggests that some components of this pathway are associated with Alzheimer's disease. Within the 50 genes, three main subgroups known to be associated with development, Alzheimer's disease, and cancer are present [HHWB05, Hei04]. *50TG* is the most heterogeneous of the five collections and is relatively well-studied [CCSS$^+$06].

115 different Interferon (IFN) genes form the *115IFN* collection. IFNs are a type of cytokine that is released by the immune system in response to viral attacks, and have recently been linked to pathways that counterbalance apoptosis (cell death) [PKP$^+$04]. All genes in the *115IFN* collection are considered IFN-stimulated genes (ISGs).

The three remaining sets of genes, denoted as *Math1*, *Mea*, and *Sey*, are all taken from microarray data obtained from mouse mutants with cerebellar defects. Each of the datasets are presently under biological examination—at this point, the genes in each of the sets are not known to be functionally related. As they pertain to current (unpublished) research, the specific genes in the cerebellar datasets will not be released, but the application of labeling methods to them is discussed.

5.2 Evaluation Techniques

As mentioned in Chapter 3, the SVD produces the mathematically optimal low-rank approximation of any matrix with respect to the Frobenius norm. While NMF can never produce a more accurate approximation than the SVD, its proximity to A relative to the SVD can be measured. Namely, the relative error, computed as

$$RE = \frac{\|A - WH\|_F - \left\|A - USV^T\right\|_F}{\|A - USV^T\|_F},$$

where both factorizations assume parameter k, can show how close the feature vectors produced by the NMF are to the optimal basis [LMA06]. In addition to relative error, the effects of the constraints with respect to convergence rates can be analyzed. A full list of the average relative errors as well as the percentage of NMF runs that converged under the various constraints is provided in Appendix B.

While measuring error norms and convergence is useful to expose mathematical properties and structural tendencies of the NMF, the ultimate goal of this application is to provide a useful labeling of a hierarchical tree from the NMF. In many cases, the "best" labeling may be provided by a suboptimal run of NMF.

Measuring recall (discussed in Chapters 2 and 4) is a quantitative way to validate "known" information within a hierarchy. To avoid confusion, the *mean average recall* (MAR) will denote the value attained when the average recall at each level is averaged

acrossed all levels.[2] For each collection, the parameter settings that provided the best labelings, both in the global and local sense, are discussed.

After applying the labeling algorithm provided by Algorithm 4.1 to the factors produced by NMF from each collection, the MAR generated was very low (under 25%) for nearly all parameter settings. Since the NMF-generated vocabulary did not overlap well with the MeSH dictionary, the NMF features were mapped into MeSH features via the procedure outlined in Algorithm 4.2, where the most dominant feature represented each document only if the corresponding weight in the H matrix was greater than 0.5.[3] Also, the top 10 MeSH headings were chosen to represent each document, and the top 100 corresponding terms were extracted to formulate each new MeSH feature vector. For each collection, the resulting MeSH feature vectors produced labelings with greatly increased MAR.

5.2.1 50TG

Relative Error and Convergence

Intuitively, as k increases, the NMF factorization should more closely approximate A. As shown in Figure 5.1, this is exactly the case. Surprisingly, however, the average of all converging NMF runs is under 10% relative error compared to the SVD, with that error tending to rise as k increases.[4] The proximity of the NMF to the SVD implies that, for this small dataset, NMF can accurately approximate the data.

[2]Here, a hierarchy level refers to all nodes that share the same distance (number of edges) from the root.

[3]Trials were run with this threshold eliminated, but this provided little effect on overall recall results.

[4]This includes runs that normalized columns of either W, H, or both, even though the normalization often increased the error.

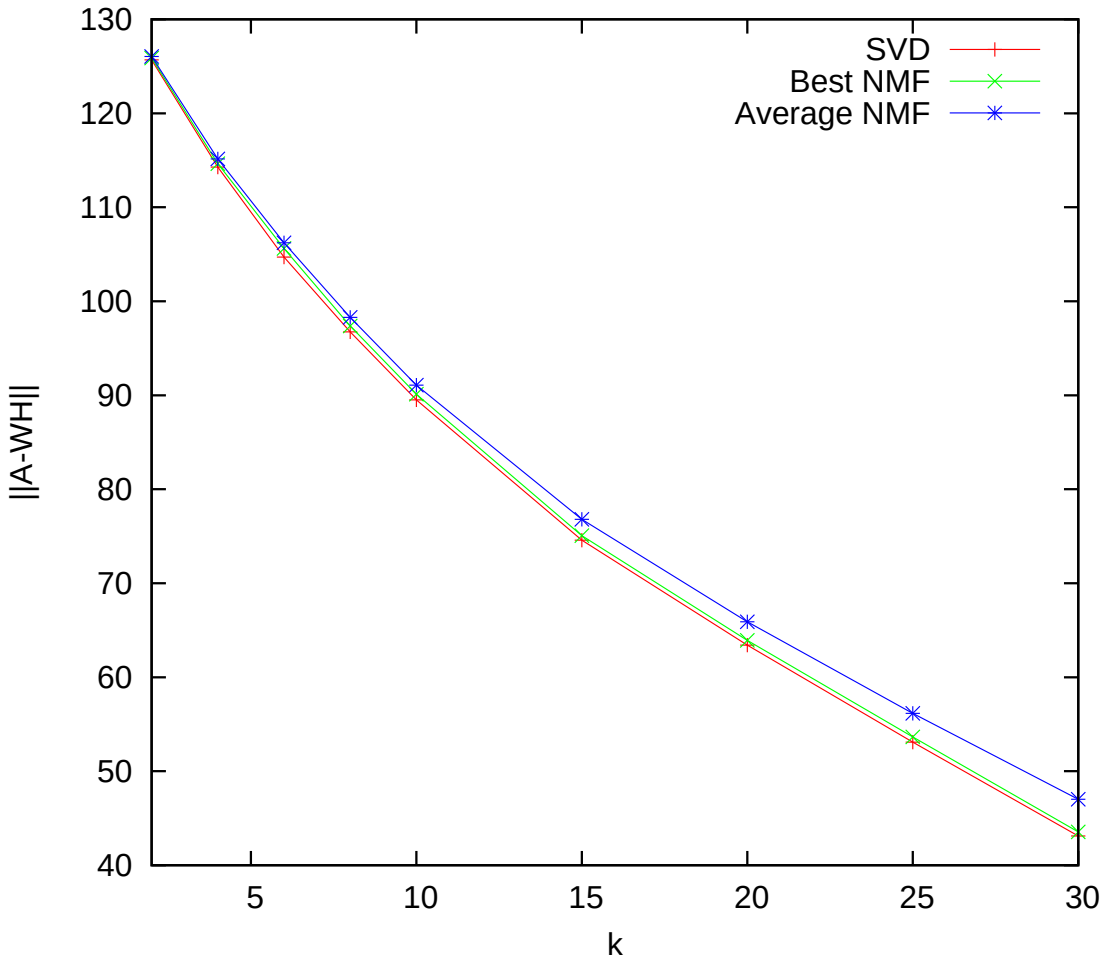


Figure 5.1: Error measures for the SVD, best NMF run, and average NMF run for the *50TG* collection.

To study the effects of convergence, one set of NMF parameters must be chosen as the baseline against which to compare. By examining the NMF with no additional constraints, the NNDSVDa initialization method consistently produces the most accurate approximation when compared to NNDSVDe, NNDSVDme, NNDSVDz, and random initialization. The relative error NNDSVDa generates is less than 1% for all tested values of $k \leq 20$, is equal to 1% for $k = 25$, and is 2.5% when $k = 30$. Unfortunately, NNDSVDa requires at least 161 and at most 331 iterations to converge.

NNDSVDe performs comparably to NNDSVDa with regard to relative error, often within a fraction of a percent. For smaller values of k, NNDSVDe takes significantly longer to converge than NNDSVDa, although the exact opposite is true for the larger value of k. NNDSVDz, on the other hand, converges much faster for smaller values of k at the cost of accuracy as the locked zero elements have an adverse effect on the best solution that can be converged upon. Not surprisingly, NNDSVDme performed comparably to NNDSVDz in many cases, however, it was able to achieve slightly more accurate approximations as the number of iterations increased.[5] Random initialization performs comparably to NNDSVDa in terms of accuracy and favorably in terms of speed for small k, but as k increases both speed and accuracy suffer. A graph illustrating the convergence rates when $k = 25$ is depicted in Figure 5.2.

In terms of actual elapsed time, the improved performance of the NNDSVD does not come without a cost. In the context of SGO, the time spent computing the initial SVD of A

[5] In fact, NNDSVDme was identical to NNDSVDz in most cases and will not be mentioned henceforth unless noteworthy behavior is observed.

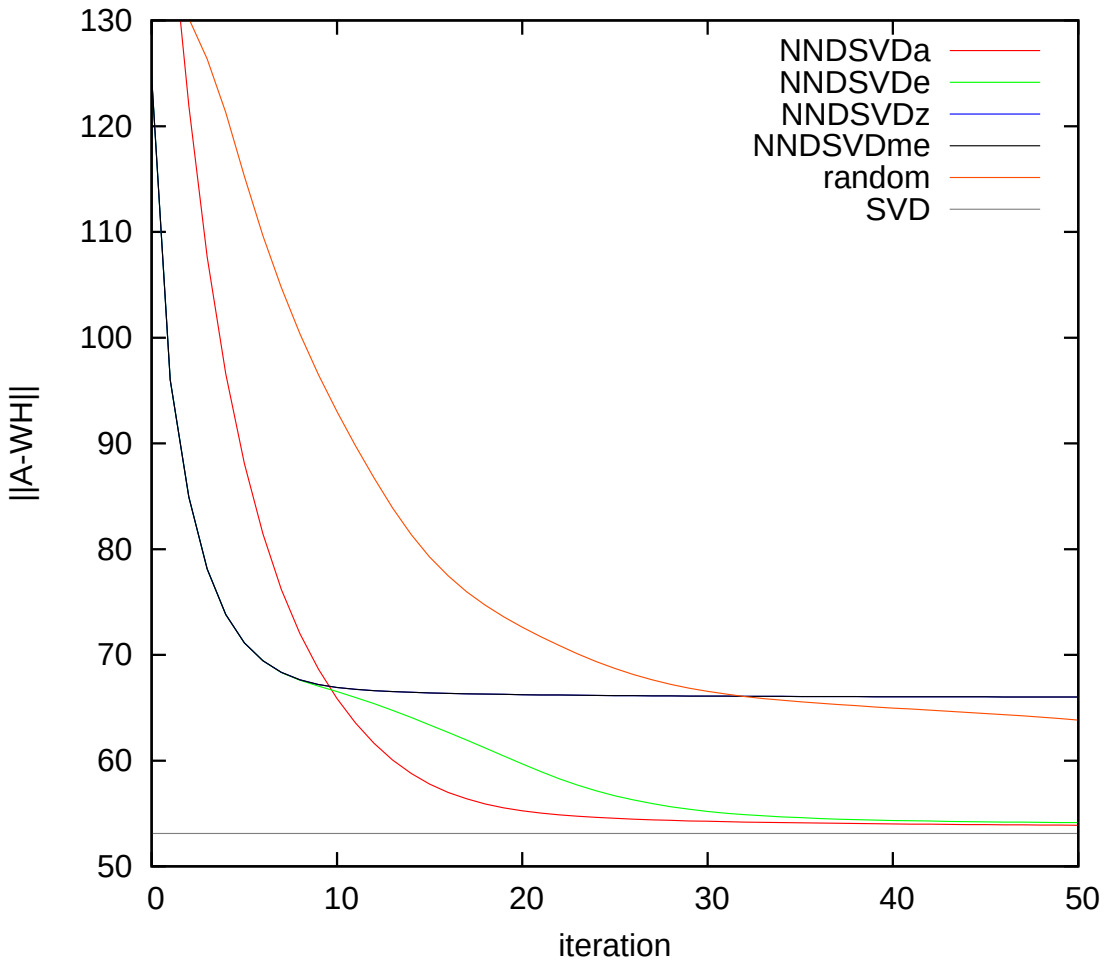


Figure 5.2: Convergence graph comparing the NNDSVDa, NNDSVDe, NNDSVDme, NNDSVDz, and best random NMF runs of the *50TG* collection for ($k = 25$).

for the first step of the NNDSVD algorithm is assumed to be zero since the SVD is needed a priori for querying purposes. The time required to complete the NNDSVD, however, is approximately 0.82 seconds per k; the initialization cost when $k = 25$ is nearly 21 seconds, while the cost for random initialization is relatively negligible.[6] Since the cost per iteration is nearly .015 seconds per k, when $k = 25$, the cost of performing the NNDSVD is approximately 55 iterations. Convergence taking into account this cost is shown in Figure 5.3.

Balancing solution quality with execution time, random initialization is usually a sufficient initialization strategy when $k < 10$, while the structure of NNDSVD improves both solution quality and overall execution time as k increases. This observation is made evident, for example, when comparing the number of iterations performed when $k = 25$. NNDSVDa can achieve 5% relative error in less than 19 iterations, while all the random initializations took at least 200 iterations to reach that accuracy, if that accuracy was even reached at all.

Assuming the NMF run for each k that incorporates the NNDSVDa initialization with no addition constraints is baseline, the effects of applying additional constraints can be examined. Applying smoothness to H had little noticeable effect as far as the overall convergence rate is concerned up to within 5% relative error. Satisfying the stationary point stopping criteria, however, was rarely attained within the 1,000 iteration limit. With $\beta = 0.1$, NNDSVDa quickly failed to converge when $k \geq 8$. Interestingly, both NNDSVDe

[6] All runs were performed on a machine running Debian Linux 3.0 with an Intel Pentium III 1 GHz processor and 256 MB memory.

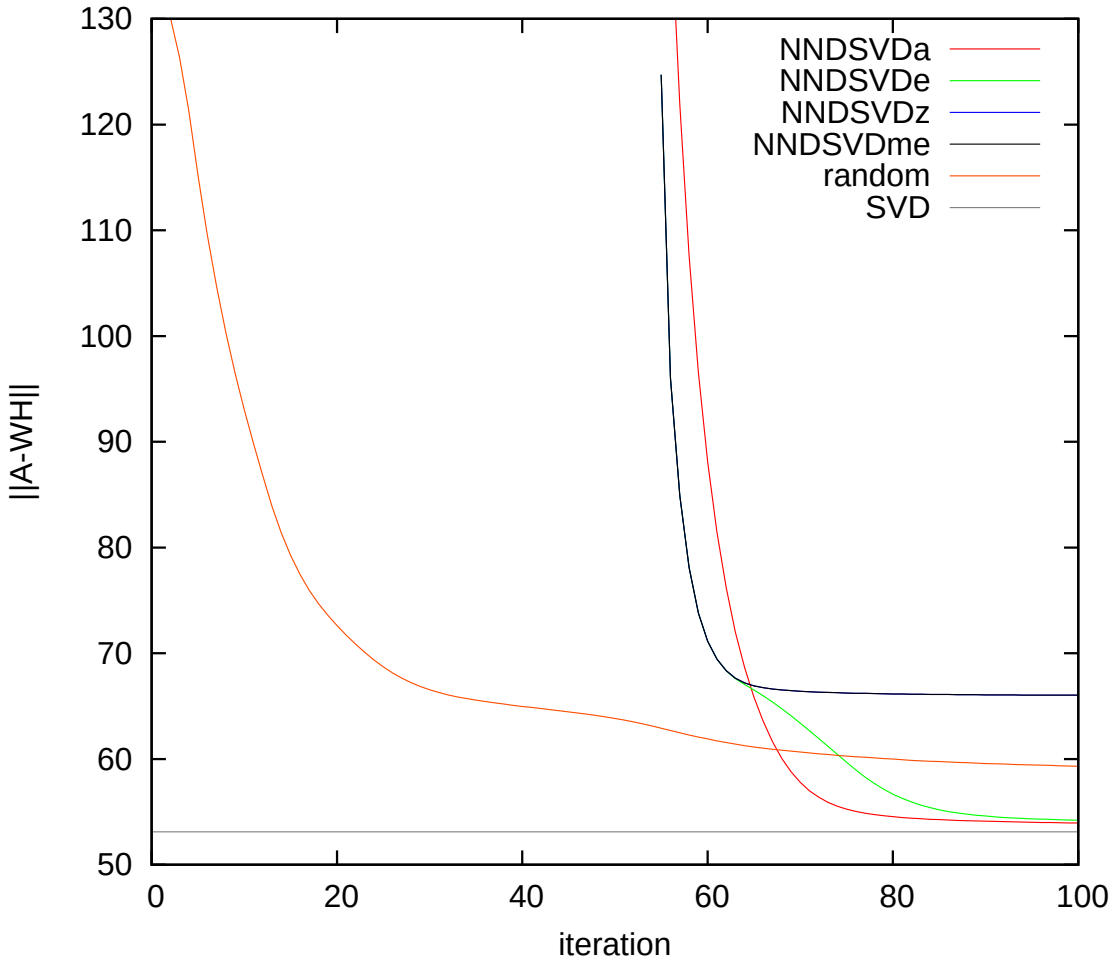


Figure 5.3: Convergence graph comparing the NNDSVDa, NNDSVDe, NNDSVDme, NNDSVDz, and best random NMF runs of the *50TG* collection for ($k = 25$) taking into account initialization time.

and NNDSVDz were able to converge for all values of β. Since NNDSVDa substitutes the initial zeros with the average value of the elements of A, placing too much importance on the norm of H (which is increased potentially significantly under NNDSVDa), overconstrains the problem. This may imply that the initial coefficient matrix H is too sparse or the number of formerly zero elements grows too large, thereby allowing the average elements substituted to carry too much weight with respect to the norm. In effect, the substitution becomes noise. By examining the sparsity of the H matrix, this hypothesis may be true as the sparsity is below 40% when $k < 8$ but climbs above 40% for the larger values of k.

Smoothing W yielded tendencies similar to smoothing H, although a stationary point was able to be reached in about the same number of iterations as the base case when α was small. As α increased, reaching that final stationary point became harder to attain within the iteration threshold.

The current implementation of the sparsity constraint is a function of two different norms with respect to a matrix. The end effect can be considered an amplification of the smoothness constraint. As such, the low convergence rate for enforcing sparsity on H should be expected, especially when the NNDSVDa initialization is applied. In fact, sparsity is so sensitive that the only value of β that yielded results that did not fail to converge was 0.001; only for small k was a stationary point reached—all other runs met the iteration limit. Table 5.2 illustrates the convergence tendencies with respect to α and β.

Enforcing sparsity on W performed similarly to enforcing sparsity on H. Smaller values of α for NNDSVDe and NNDSVDz once again attained the maximum number of

Table 5.2: Number of iterations required to reach specific percentages of relative error. These values reflect runs where $k = 25$ and the NNDSVDe initialization was used. Values in parentheses indicate the point where NMF was terminated due to increasing objective function. All other runs enforcing sparsity terminated after 1 or 2 iterations.

			Iterations Required To Reach Relative Error							
Constraint	Parameter (α or β)	Sparsity	Stationary Point	1%	5%	10%	15%	20%	25%	50%
none			249		28	22	19	15	11	3
sparseW	0.001	0.1	(3)							
sparseW	0.001	0.25	1000		26	21	17	14	10	3
sparseW	0.001	0.5	1000		27	21	18	15	11	3
sparseW	0.001	0.75	(18)					15	11	3
sparseW	0.001	0.9	(6)							3
smoothW	0.001		1000		28	22	19	15	11	3
smoothW	0.01		1000		29	23	19	15	11	3
smoothW	0.1		1000		30	24	20	16	12	3
sparseH	0.001	0.1	(2)							
sparseH	0.001	0.25	1000		37	28	24	19	15	4
sparseH	0.001	0.5	1000		36	28	23	19	14	4
sparseH	0.001	0.75	(51)		31	24	20	16	12	3
sparseH	0.001	0.9	(7)							3
smoothH	0.001		250		28	22	19	15	11	3
smoothH	0.01		253		28	22	19	15	11	3
smoothH	0.1		1000		28	22	19	15	11	3

iterations. Interestingly, sparsity values of 0.5 and 0.25 generated the most stable results, with 0.5 yielding 1,000 iterations for all values of k. Figures 5.4 and 5.5 show the effect the various constraints have on the $\|W\|$ and $\|H\|$, respectively, while Figure 5.6 shows the relative consistency of the corresponding approximations.[7]

If structured initialization is ignored and only random initialization runs examined, then several observations can be made. First, very few combinations of parameter settings yielded converging results when sparsity was enforced on W. Second, enforcing sparsity, particularly on H, while more likely not to converge, was also more likely to reduce its relative error quickly with respect to the number of iterations. Third, smoothing either W

[7]To include sparsity in these figures, the NNDSVDe initialization was used.

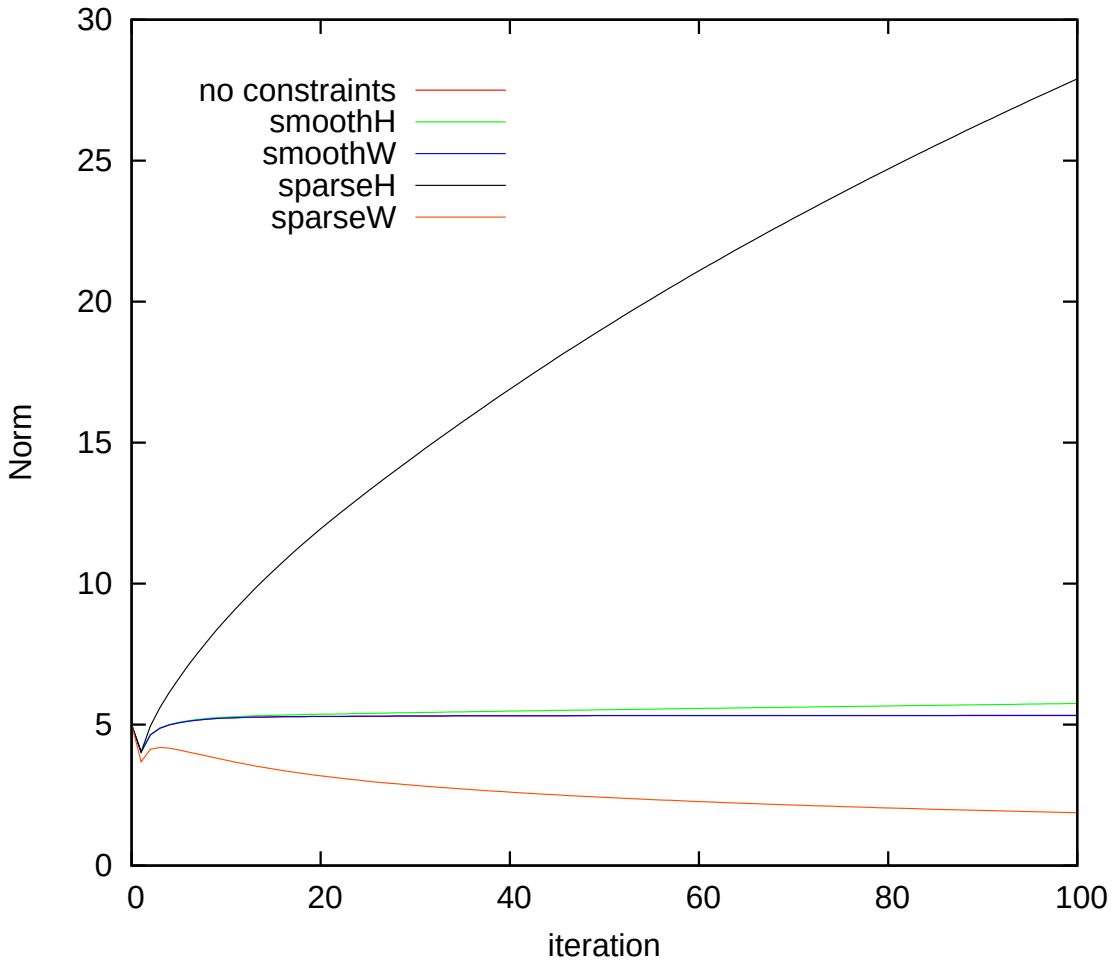


Figure 5.4: $\|W\|_F$ after each iteration.

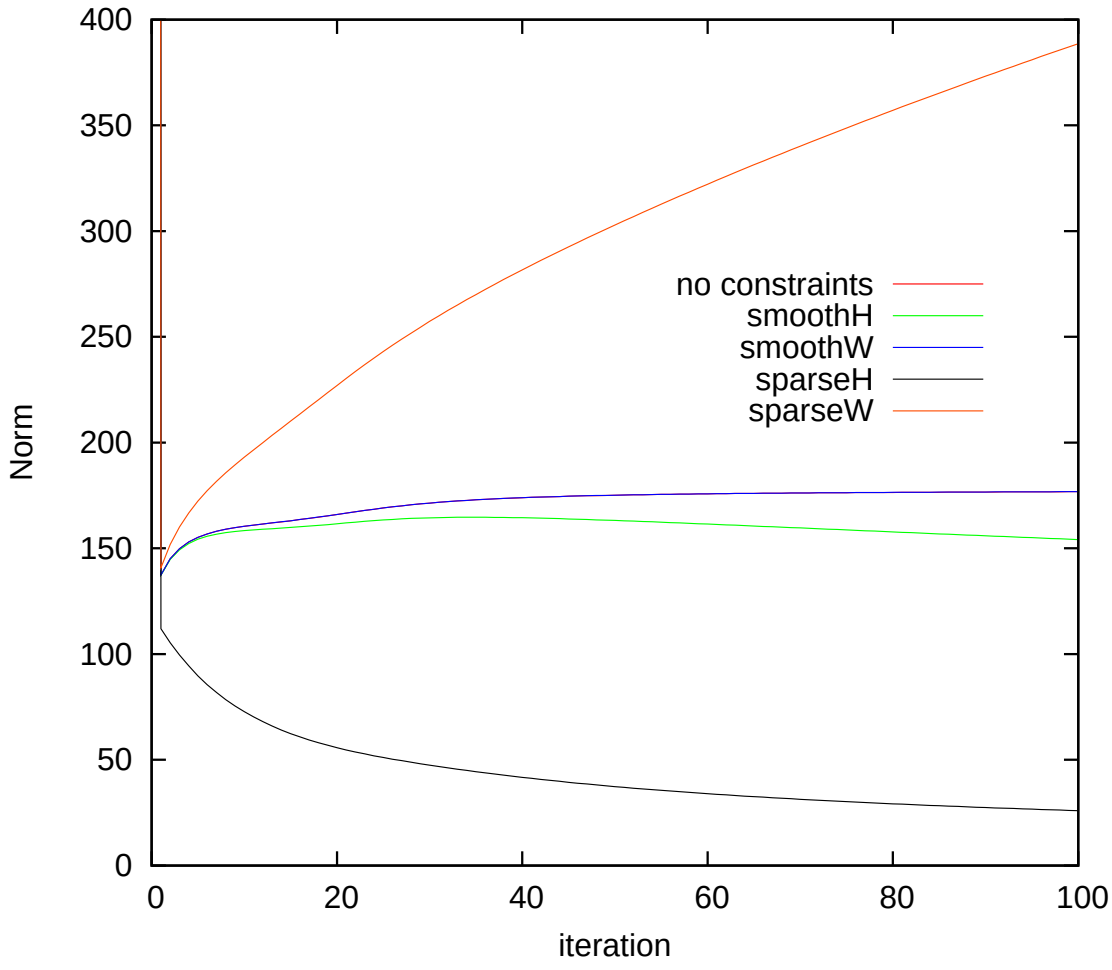


Figure 5.5: $\|H\|_F$ after each iteration.

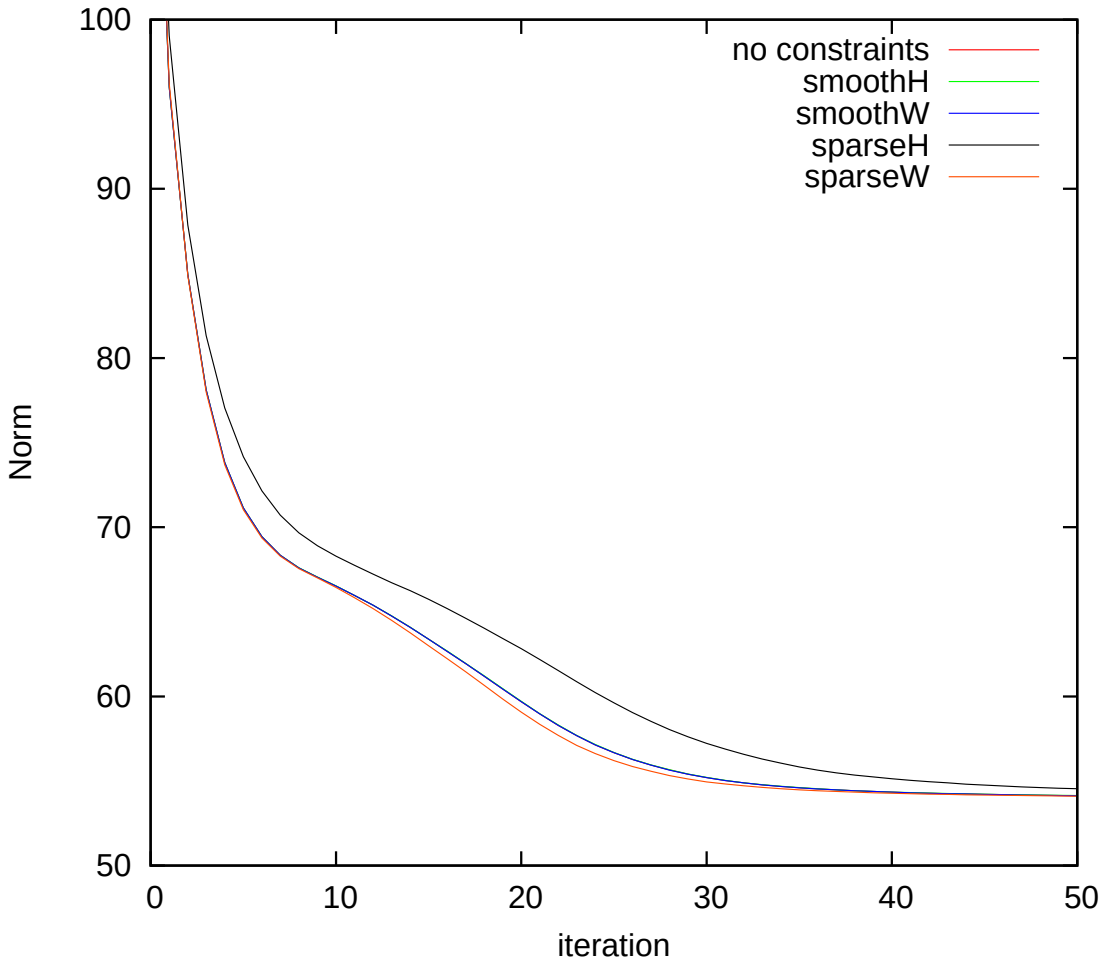


Figure 5.6: $\|A - WH\|_F$ after each iteration.

or H tended to speed convergence to within 5% relative error, and the number of iterations required to reach a stationary point increased as the smoothing parameter increased.

As expected, smoothing H tended to decrease the magnitude of the values within H at the cost of increasing the values in W. Smoothing W produced a similar (and opposite) effect, although not to the same degree. If sparsity is enforced on H, the magnitude of the values within H are greatly reduced and correspondingly compensated for with adjustments in W. Similarly, sparsity on W shrinks the relative magnitude of W at the expense of H, and unlike its smoothing counterpart, the values attained differ greatly from the unconstrained case. Enforcing smoothness or sparsity on H causes different tokens to be prominent within each feature and affects the labelings produced.

When the vocabulary size is reduced by applying a 0.8 global threshold, smaller relative error is expected since the vocabulary size is significantly decreased. Figure 5.7 shows how relative error increases with k, although the relative error for each k is less than its *50TG* counterpart.

As far as initialization strategies are concerned, NNDSVDa remains the best in terms of relative error, but only for $k \leq 15$. For $k \geq 20$, NNDSVDe outperforms NNDSVDa in terms of both relative error and number of iterations required to converge. In many cases, the NNDSVDe achieved relative error of 1% in as many or fewer iterations as NNDSVDa, so the NNDSVDe will be considered the baseline against which to compare the effects of constraints on convergence speed for the *50TG.8* collection. In general, applying smoothing constraints shows consequences similar to their *50TG* counterparts. In

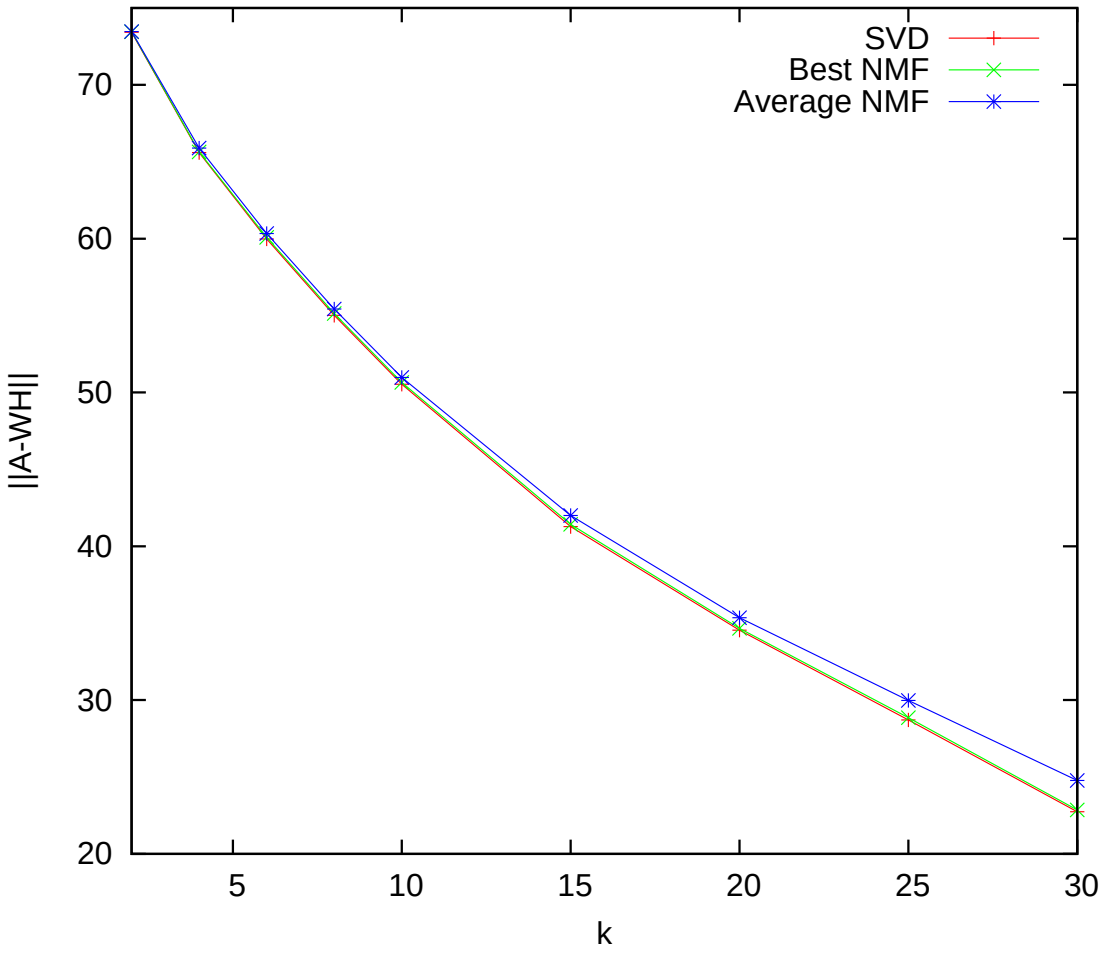


Figure 5.7: Error measures for the SVD, best NMF run, and average NMF run for the *50TG.8* collection.

particular, NNDSVDa fails to converge when smoothing H if $\beta = 0.1$ and $k > 6$, which corresponds to 30% original sparsity in H. Additionally, convergence is not achieved if $\beta = 0.01$ and $k = 30$ when smoothing H, which corresponds to a 45% original sparsity in H. Also similar to its *50TG* counterpart, applying sparsity to either factor increases the number of iterations required to reach a stationary point, generally only converges for smaller values of α or β, and greatly reduces the norm of the corresponding constrained factor at the cost of increasing the magnitude of the other. Since the matrix associated with the *50TG.8* collection is smaller than the *50TG* matrix, some small values of k were able to converge to a stationary point within the 1,000 iteration limit.

Labeling Recall

With regard to the accuracy of the labelings, several trends exist. As k increases, the achieved MAR increases as well. This behavior could be predicted since increasing the number of features also increases the size of the effective labeling vocabulary, thus enabling a more robust labeling. When $k = 25$, the average MAR across all runs is approximately 68%.

Since the NNDSVDa initialization provided the best convergence properties, it shall be used as a baseline against which to compare.[8] In terms of MAR, NNDSVDa produced below average results, with both NNDSVDe and NNDSVDz consistently outperforming NNDSVDa for most values of k; NNDSVDe and NNDSVDz attained similar MAR values

[8] If k is not specified, assume $k = 25$.

as depicted in Figure 5.8. The recall of the baseline case using NNDSVDa and $k = 25$ depicted by node level is shown in Figure 5.9.

The 11 levels of nodes can be broken into thirds to analyze the accuracy of a labeling within a depth region of the tree. The MAR for NNDSVDa for each of the thirds is approximately 58%, 63%, and 54%, respectively. With respect to the topmost third of the tree, any constraint applied to any NNDSVD initialization other than smoothing W applied to NNDSVDa provided an improvement over the 58% MAR. In all cases, the resulting MAR was at least 75%. NNDSVDa performed slightly below average over the middle third at 63%. Overall, nearly any constraint improved or matched recall over the base case over all thirds with the exception that enforcing sparsity on H underperformed NNDSVDa in the bottom third of the tree; all other constraints achieved at least 54% MAR for the bottom third.

With respect to different values of k, similar tendencies exist over all thirds. NNDSVDa is among the worst in terms of MAR with the exception that it does well in the topmost third when k is either 2 or 4. There was no discernable advantage when comparing NNDSVD initialization to its random counterpart. Overall, the best NNDSVD (and hence reproducible) MAR was achieved when enforcing sparsity on H with $\beta = 0.001$, sparseness parameter 0.5 or 0.75, and $k = 30$ (shown in Figure 5.10). The nearly 78% MAR achieved was reached when NNDSVDz, NNDSVDe, or NNDSVDme was used.

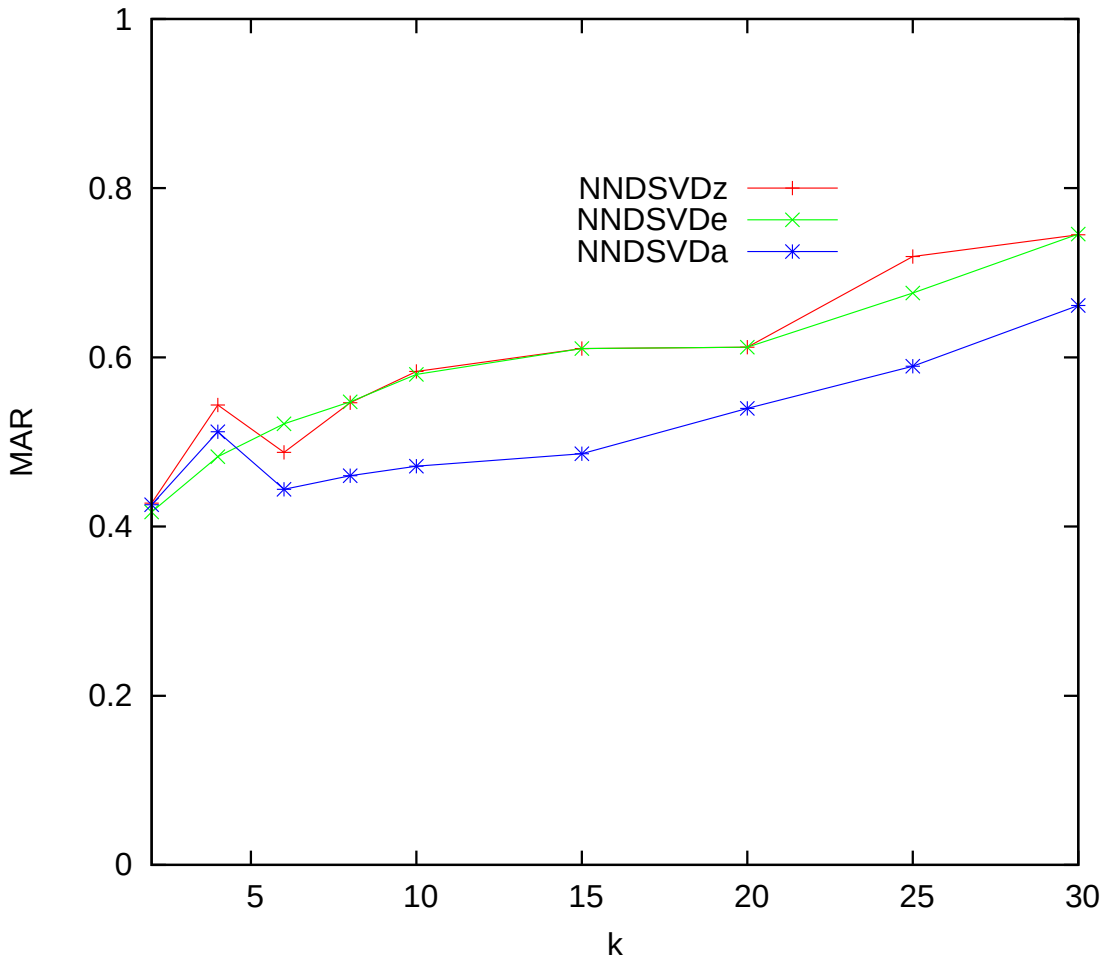


Figure 5.8: MAR as a function of k under the various NNDSVD initialization schemes with no constraints for the *50TG* collection.

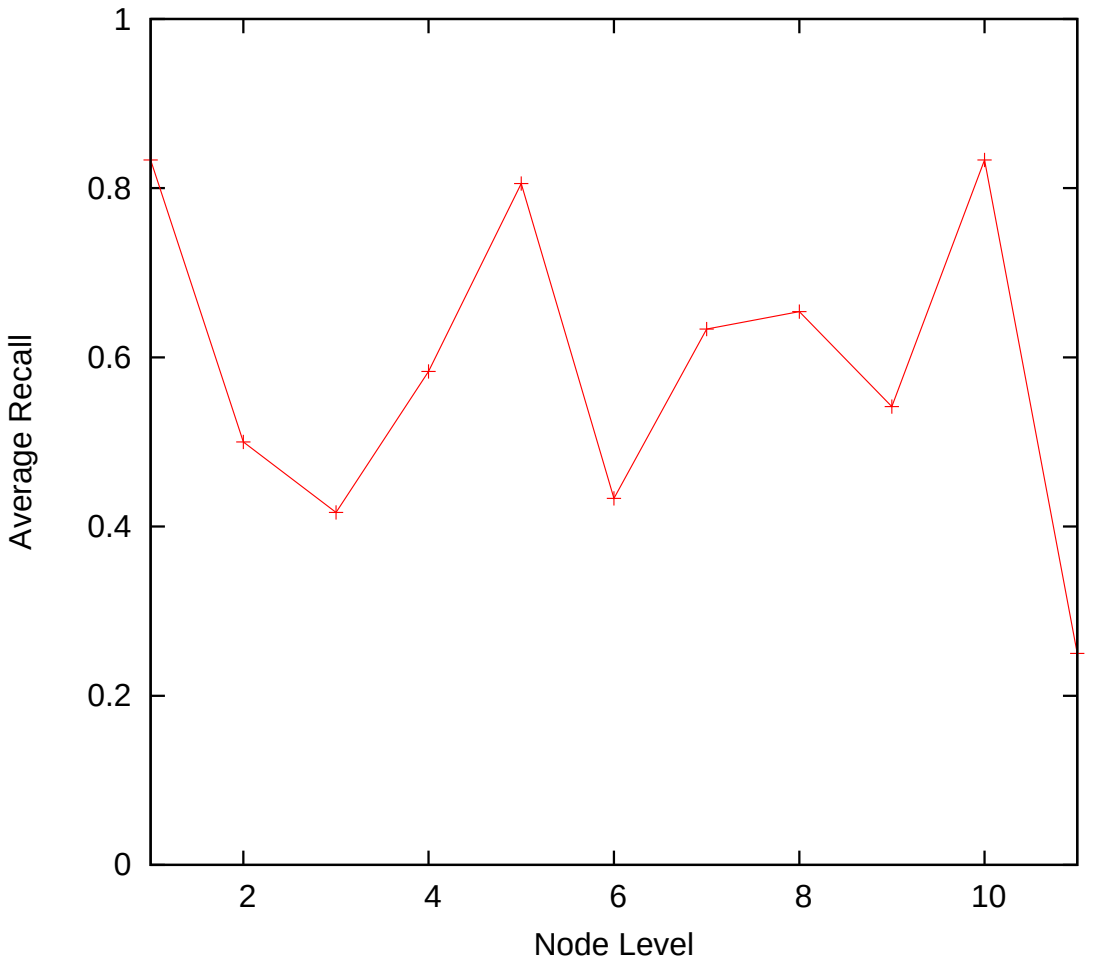


Figure 5.9: Recall as a function of node level for the NNDSVDa initialization on the *50TG* collection. The achieved MAR is 58.95%.

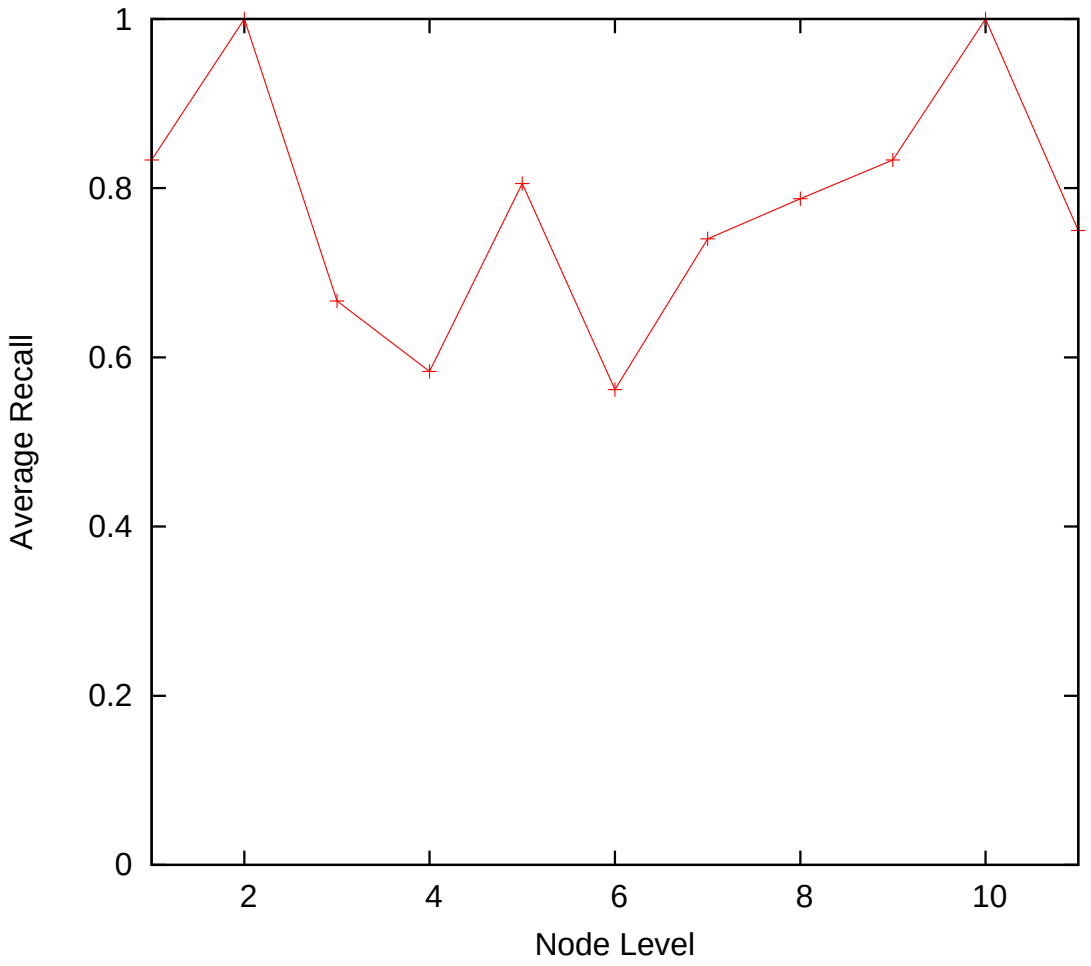


Figure 5.10: Best recall achieved by the NNDSVD initialization for the *50TG* collection. The achieved MAR is 77.83%.

5.2.2 115IFN

Relative Error and Convergence

Similar to the *50TG* collection, the NNDSVDa initialization strategy is the most robust in terms of fewest iterations required to reach 1% relative error for the *115IFN* collection. The larger size of the dataset, however, becomes problematic as computing the NNDSVD is equivalent to performing approximately 249 NMF iterations. The relative error of the various initializations is given in Figure 5.11. Assuming that an NNDSVD is performed, however, faster convergence rates can be expected relative to its random counterpart for larger k.

Interestingly, in some instances, although the time required to converge did not improve, smoothing H reduced the number of iterations required to reach small relative errors. This occurred most often for larger k and for smaller smoothing parameters. As with the *50TG* set, a value of 0.1 proved too stringent and caused NMF to fail to converge. When enforcing sparsity on H, sparsity values of 0.5 and 0.75 proved to be the most successful for small β. Enforcing either constraint on W generated results consistent with those observed with the *50TG* collection. By reducing the vocabulary size to the *115IFN.8* collection, NNDSVDa remained the best initialization strategy except when k was 8, 10, or 15, when NNDSVDe was superior. Enforcing sparsity on either the *115IFN* and *115IFN.8* seemed to be sensitive to the initialization method, as both collections failed to converge for all random initializations when $k \geq 10$.

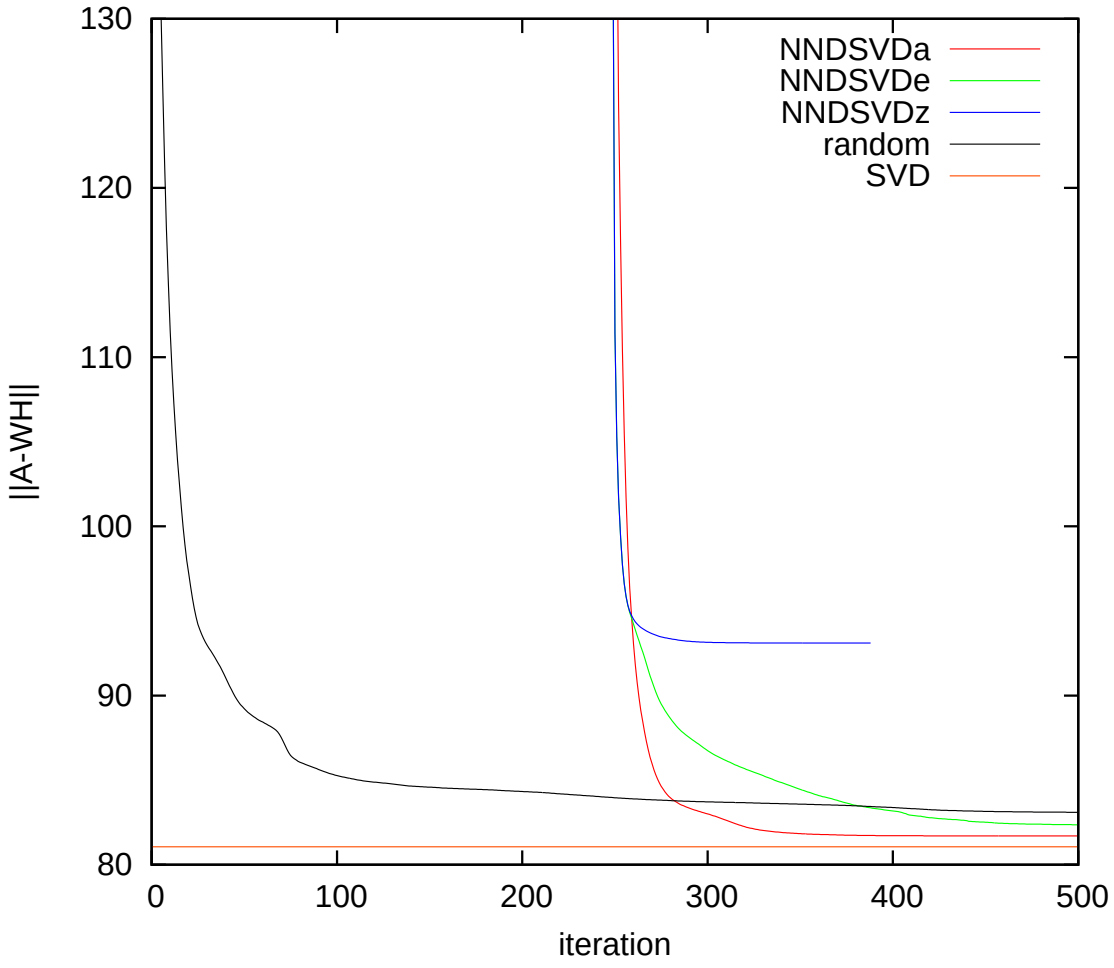


Figure 5.11: Convergence graph comparing the NNDSVDa, NNDSVDe, NNDSVDz, and best random NMF runs of the *115IFN* collection for ($k = 25$) taking into account initialization time.

Labeling Recall

The MAR achieved by the *115IFN* collection was, on average, comparable to the *50TG* collection, although the best cases were not nearly as good as the *50TG* collection. As with *50TG*, MAR increased as k increased. Unlike *50TG*, however, the NNDSVDa initialization produced MAR competitive with the other NNDSVD strategies. The NNDSVDa enforcing sparsity on W with $\alpha = 0.001$, $k = 30$, and sparsity parameter 0.5 produced the best MAR for structured initialization runs, and its recall is depicted in Figure 5.12. Broken into thirds, the MAR is 20%, 50%, and 79%.

5.2.3 Cerebellar Data Sets

Relative Error and Convergence

All three of the cerebellar data sets contained a slightly smaller number of documents and approximately half the number of terms of the *50TG* set. Consequently, the actual error achieved by the SVD was extremely low especially for large k, and the resulting relative errors of the NMF runs grew at a higher rate as k increased. In general, NNDSVDa was the best overall initialization strategy for most k in terms of convergence speed and accuracy, although NNDSVDe tended to outperform for larger values of k for the *Math1* and *Sey* collections. The relative cost of initialization in terms of number of iterations was approximately 59, 59, and 37 iterations for the *Math1*, *Mea*, and *Sey* collections, respectively, and in each case the overall speed of an NNDSVD-initialized NMF was much faster than its randomized counterpart. With regard to the effect of additional constraints, nearly all

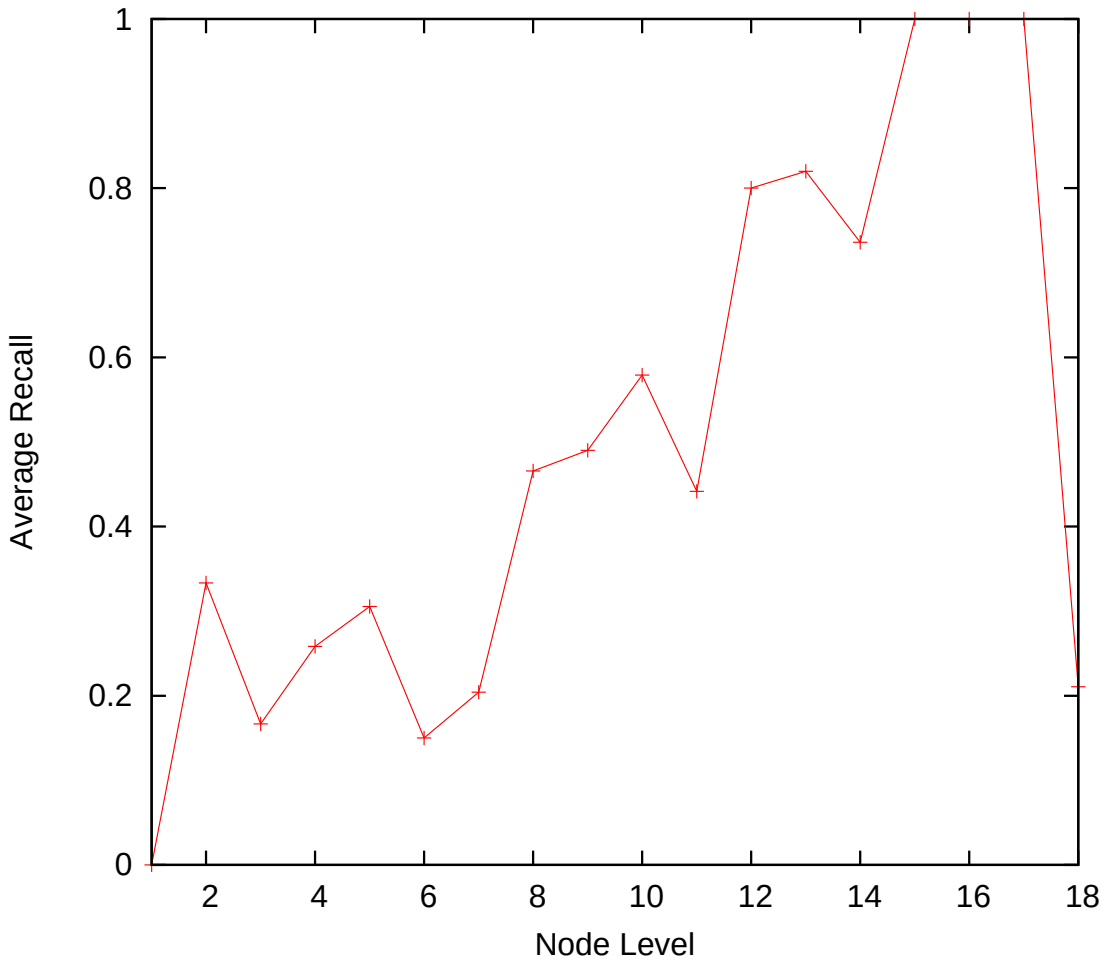


Figure 5.12: Best recall achieved by the NNDSVD initialization for the *115IFN* collection. The achieved MAR is 49.79%.

the tendencies observed with the *50TG* collection are consistent with those seen with the cerebellar data sets.

Labeling Recall

Best performance in terms of MAR was achieved for higher values of k. With respect to initialization, NNDSVD performed better than its random counterpart for the *Mea* collection, although random initialization was on par with NNDSVD for both the *Math1* and *Sey* collections. As with the *50TG* collection, NNDSVDa tended to produce lower MAR than the other three NNDSVD methods. Overall, the highest MAR values achieved for each of the *Math1*, *Mea*, and *Sey* datasets were 62%, 57%, and 81%, respectively.

5.2.4 General Observations

While comparing NMF runs over all five collections, several trends can be observed, both with respect to mathematical properties and recall tendencies. First, and as expected, as k increases, the approximation achieved by the SVD with respect to A is more accurate; the NMF can provide a relatively close approximation to A in most cases, but the error also increases with k. Second, NNDSVDa provides the fastest convergence in terms of number of iterations to the closest approximations. Third, enforcing either sparsity or smoothing on H tends to have a more noticeable effect on the norm of H than performing the same operation on W. Also, enforcing H constraints on a randomized seed would frequently speed convergence to within 5% relative error, although it would rarely reach a stationary

point. Of course, this observation may also be related to quality of the random seed generated. Unfortunately, convergence was a major issue, especially with sparsity. Sparsity was extremely sensitive to the initial seed, and usually only small values of α and β led to converging solutions. Extreme sparseness values of 0.1 or 0.9 rarely converged, while 0.5 proved to be the most successful. Finally, to generate relatively "good" approximation error (within 5%), about 20–40 iterations are recommended using either NNDSVDa or NNDSVDe initialization with no additional constraints when k is reasonably large (about half the number of documents). For smaller k, performing approximately 25 iterations under random initialization will usually accomplish 5% relative error, with the number of iterations required decreasing as k decreases.

Frequently, the best labelings did not come from the most accurate approximations. Overall, more accurate labelings resulted from higher values of k because more feature vectors increased the vocabulary size of the labeling dictionary. Generally speaking, the NNDSVDe, NNDSVDme, and NNDSVDz schemes outperformed the NNDSVDa initialization. Enforcing additional constraints on either W or H with small parameter tended to increase the achieved MAR slightly for many of the NNDSVD runs. Overall, the accuracy of the labelings appeared to be more a function of k and the initial seed rather than the constraints applied.

Chapter 6

Conclusion

SGO and other text mining systems have become more popular in biological research and applications. As different methods become available, they must be tailored to be useful in their application domain, and attempts must be made to explain their functionality in different terminology. In the case of SGO, the negativity imposed by the SVD creates interpretability issues. As a result, the use of NMF was explored, and the application of NMF to labeling hierarchical trees was examined on five different data sets. Also, automated methods were suggested to expand existing annotations, classify groups of genes, and evaluate those classifications. Currently, these three tasks are frequently performed manually [CCSS$^+$06].

Much research is being performed concerning the NMF, and this work examines three methods based on the MM update [LS99]. Many other NMF variations exist and more are being developed, so their application to the biological realm should be studied. For

example, [SBPP06] proposes a hybrid least squares approach called GD-CLS to solve NMF and overcomes the problem of "locking" zeroed elements encountered by MM, [PMCK+06] proposes nonsmooth NMF as an alternative method to incorporate sparseness, and [DLPP06] proposes an NMF technique that generates three factor matrices and has shown promising clustering results. NMF has been applied to microarray data [BTGM04], but efforts need to be made to combine the text information with microarray data; some variation of tensor factorization could possibly show how relationships change over time [CZC+07].

With respect to the labeling method, MeSH terms are assigned to genes, but the MeSH hierarchy is ignored. Developing methods to incorporate the information inherent in that hierarchy could help produce more accurate gene classifications. The trees themselves are built using simplifying assumptions—extending trees into hierarchical graphs has more real applicability, although algorithm complexity may be prohibitive at the moment. Also, methods such as natural language processing (NLP) techniques could be used to examine sentence structure. Consequently, both the edges as well as nodes of the tree could be labeled where nodes show the function of a gene and edges further specifies a gene's effect on that function. Of course, as data become more complex, visualization techniques are another issue that this dissertation has ignored.

Regardless of the techniques employed, one of the issues that will always be prevalent regarding biological data is that of quality versus quantity. Inherently related to this problem is the establishment of standards within the field especially as they pertain to hi-

erarchical data. Efforts such as Gene Ontology (GO) are being built and refined [Con00], but standard data sets against which to compare results and clearly defined and accepted evaluation measures could facilitate clear comparison between differing methods.

In the case of SGO, developing methods to derive "known" data is a major issue (even GO does not produce a "gold standard" hierarchy given a set of genes). Access to more data and to other hierarchies would help test the robustness of the method, but that remains one of the problems inherent in the field. As discussed in Chapter 5, approximations that are more mathematically optimal do not always produce the "best" labeling. Often, factorizations provided by the NMF can be deemed "good enough," and the final evaluation will remain subjective. In the end, if automated approaches can approximate that subjectivity, then greater understanding of more data will result.

Bibliography

Bibliography

[BB99] M.W. Berry and M. Browne. *Understanding Search Engines: Mathematical Modeling and Text Retrieval*. SIAM, Philadelphia, PA, 1999.

[BBL+06] M.W. Berry, M. Browne, A.N. Langville, V.P. Pauca, and R.J. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. 2006. Preprint.

[BFW92] H. L. Bodlaender, M. R. Fellows, and T. J. Warnow. Two strikes against perfect phylogeny. In *Proceedings of the 19th International Colloqium on Automata, Languages and Programming*, pages 273–283, Berlin, 1992. Springer Verlag, Lecture Notes in Computer Science 623.

[BG05] C. Boutsidis and E. Gallopoulos. On SVD-based initialization for nonnegative matrix factorization. Technical report, Tech Report HPCLAB-SCG-6/08-05, University of Patras, Patras, Greece, 2005.

[BSH00] W.J. Bruno, N.D. Socci, and A.L. Halpern. Weight neighbor joining: A likelihood–based approach to distance–based phylogeny reconstruction.

Molecular Biology and Evolution, 17(1):189–197, 2000.

[BST06] Z. Barutcuoglu, R.E. Schapire, and O.G. Troyanskaya. Hierarchical multi–label prediction of gene function. *Bioinformatics*, 22(7):830–836, 2006.

[BTGM04] J.P. Brunet, P. Tamayo, T.R. Golub, and J.P. Mesirov. Metagenes and molecular pattern discovery using matrix factorizations. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 101, pages 4164–4169. National Academy of Sciences, 2004.

[BYRN99] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison–Wesley, New York, 1999.

[CCSS+06] Monica Chagoyen, Pedro Carmona-Saez, Hagit Shatkay, Jose M. Carazo, and Alberto Pascual-Montano. Discovering semantic features in the literature: a foundation for building functional associations. *BMC Bioinformatics*, 7(41), 2006.

[Con00] The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[CZC+07] A. Cichocki, R. Zdunek, S. Choi, R. Plemmons, and S. Amari. Novel multilayer nonnegative tensor factorization with sparsity constraints. In *Proceedings of the 8th International Conference on Adaptive and Natural Computing Algorithms*, Warsaw, Poland, 2007. Preprint.

[Dam95] M. Damashek. Gauging similarity with n-grams: Language-independent categorization of text. *Science*, 267(5199):843–848, 1995.

[DDL+90] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[DFG01] I. Dhillon, J. Fan, and Y. Guan. Efficient clustering of very large document collections. In R. Grossman, G. Kamath, and R. Naburu, editors, *Data Mining for Scientific and Engineering Applications*. Kluwer Academic Publishers, 2001.

[DG02] R. Desper and O. Gascuel. Fast and accurate phylogeny reconstruction algorithms based on the minimum-evolution principle. *Journal of Computational Biology*, 19(5):687–705, 2002.

[DKM06] P. Drineas, R. Kannan, and M. Mahoney. Fast monte carlo algorithms for matrices iii: Computing a compressed approximate matrix decomposition, 2006.

[DLPP06] C. Ding, T. Li, W. Peng, and H. Park. Orthogonal nonnegative matrix tri-factorizations for clustering. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 126–135, Philadelphia, PA, 2006. ACM Press.

[EY36] C. Eckert and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1:211–218, 1936.

[Fel82] J. Felsenstein. Numerical methods for inferring evolutionary trees. *The Quarterly Review of Biology*, 57(4):379–404, 1982.

[Fit71] W.M. Fitch. Toward defining the course of evolution: Minimum change for a specific tree topology. *Systematic Zoology*, 20(4):406–416, 1971.

[FM67] W.M. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 155(3760):279–284, 1967.

[FR83] M.E. Funk and C.A. Reid. Indexing consistency in MEDLINE. *Bulletin of the Medical Library Association*, 71(2):176–183, 1983.

[Gas97] O. Gascuel. BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data. *Molecular Biology and Evolution*, 14(7):685–695, 1997.

[GJW82] M.R. Garey, D.S. Johnson, and H.S. Witsenhausen. The complexity of the generalized lloyd-max problem. *IEEE Transactions on Information Theory*, 28(2):255–256, 1982.

[GL96] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 3rd edition, 1996.

[HBC+95] D. Harman, C. Buckley, J. Callan, S. Dumais, D. Lewis, S. Robertson, A. Smeaton, K. Jones, R. Tong, G. Salton, and M. Damashek. Performance of text retrieval systems. *Science*, 268(5216):1417–1420, 1995.

[Hei04] K.E. Heinrich. Finding functional gene relationships using the Semantic Gene Organizer (SGO). Master's thesis, Department of Computer Science, University of Tennessee, 2004.

[Hei06] K.E. Heinrich. Clutering: Partitional algorithms. In M.W. Berry and M. Browne, editors, *Lecture Notes in Data Mining*, pages 121–132, Hackensack, NJ, 2006. World Scientific.

[HHWB05] R. Homayouni, K. Heinrich, L. Wei, and M.W. Berry. Gene clustering by latent semantic indexing of MEDLINE abstracts. *Bioinformatics*, 21(1):104–115, 2005.

[Hoy02] P.O. Hoyer. Non-negative sparse coding. In *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing*, Martigny, Switzerland, 2002.

[Hoy04] P.O. Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, 5:1457–1469, 2004.

[Hul96] D.A. Hull. Stemming algorithms: A case study for detailed evaluation. *Journal of the American Society of Information Science*, 47(1):70–84, 1996.

[JLKH01] T.K. Jenssen, A. Lægreid, J. Komorowski, and E. Hovig. A literature network of human genes for high-throughput analysis of gene expression. *Nature Genetics*, 28:21–28, 2001.

[JP04] N.C. Jones and P.A. Pevzner. *An Introduction to Bioinformatics Algorithms*. MIT Press, Cambridge, Massachusetts, 2004.

[Kir05] S. Kiritchenko. *Hierarchical Text Categorization and Its Applications to Bioinformatics*. PhD thesis, University of Ottawa, 2005.

[KMF05] S. Kiritchenko, S. Matwin, and A.F. Famili. Functional annotation of genes using hierarchical text categorization. In *Proceedings of the BioLINK SIG: Linking Literature, Information and Knowledge for Biology, a Joint Meeting of the ISMB BioLINK Special Interest Group on Text Data Minging and the ACL Workshop on Linking Biological Literature, Ontologies and Databases: Mining Biological Semantics (held at ISMB-05)*, 2005.

[KPR98] J. Kleinberg, C.H. Papadimitriou, and P. Raghavan. A microeconomic view of data mining. *Data Mining and Knowledge Discovery*, 2(4):311–324, 1998.

[KSZ71] K. Kidd and L. Sgaramella-Zonta. Phylogenetic analysis: Concepts and methods. *American Journal of Human Genetics*, 23:235–252, 1971.

[LH74] C.L. Lawson and R.J. Hanson. *Solving Least Squares Problems*. Prentice-Hall, Englewood Cliffs, NJ, 1974.

[LMA06] A. Langville, C. Meyer, and R. Albright. Initializations for the nonnegative matrix factorization. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Philadelphia, Pennsylvania, 2006.

[LS99] D.D. Lee and H.S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

[LS01] D.D. Lee and H.S. Seung. Algorithms for non-negative matrix factorization. *Advances in Neural Information Processing Systems*, 13:556–562, 2001.

[MOPT05] D. Maglott, J. Ostell, K. Pruitt, and T. Tatusova. Entrez gene: gene-centered information at NCBI. *Nucleic Acids Research*, 33 (Database Issue):D54–D58, 2005.

[NLM] NLM fact sheets. www.nlm.nih.gov/pubs/factsheets/factsheets.html, accessed January, 2007.

[PKP+04] L.M. Pfeffer, J.G. Kim, S.R. Pfeffer, D.J. Carrigan, D.P. Baker, L. Wei, and R. Homayouni. Role of nuclear factor-κb in the antiviral action of interferon and interferon-regulated gene expression. *Journal of Biological Chemistry*, 279(30):31304–31311, 2004.

[PMCK+06] A. Pascual-Montano, J.M. Carazo, K. Kochi, D. Lehmann, and R.D. Pascual-Marqui. Nonsmooth nonnegative matrix factorization (nsNMF). *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(3):403–415, 2006.

[Por80] M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[PPA07] V.P. Pauca, R.J. Plemmons, and K. Abercromby. Nonnegative matrix factorization with physical constraints for spectral unmixing, 2007. in preparation.

[PPPG04] J. Piper, V.P. Pauca, R.J. Plemmons, and M. Giffin. Object characterization from spectral data using nonnegative factorization and information theory. In *Proceedings of the AMOS Technical Conference*, Maui, Hawaii, 2004.

[PT94] P. Paatero and U. Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 1994.

[Rob71] D.F. Robinson. Comparison of labeled trees with valency three. *Journal of Combinatorial Theory (B)*, 11:105–119, 1971.

[San75] D. Sankoff. Minimal mutation trees of sequences. *SIAM Journal of Applied Mathematics*, 28:35–42, 1975.

[San05] M. Sandler. On the use of linear programming for unsupervised text classification. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 256–264, Chicago, Illinois, 2005.

[SB03] G.W. Stuart and M.W. Berry. A comprehensive whole genome bacterial phylogeny using correlated peptide motifs defined in a high dimensional vector space. *Journal of Bioinformatics and Computational Biology*, 1(3):475–493, 2003.

[SBPP06] F. Shahnaz, M.W. Berry, V.P. Pauca, and R.J. Plemmons. Document clustering using nonnegative matrix factorization. *Information Processing and Management*, 42(2):373–386, 2006.

[Sha01] R. Shamir. Algorithms for molecular biology: Phylogenetics and phylogenetic trees, December 27, 2001. Lecture Notes.

[SN87] N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.

[vRRP80] C.J. van Rijsbergen, S.E. Robertson, and M.F. Porter. *New Models in Probabilistic Information Retrieval*. British Library, London, 1980.

[WCD03] S. Wild, J. Curry, and A. Dougherty. Motivating non-negative matrix factorizations. In *Proceedings of the Eighth SIAM Conference on Applied Linear Algebra*, Williamsburg, Virginia, 2003.

[WCD04] S. Wild, J. Curry, and A. Dougherty. Improving non-negative matrix factorizations through structured initialization. *Pattern Recognition*, 37:2217–2232, 2004.

Appendices

Appendix A

Genes Used in Test Data Set

The genes comprising each data set along with other associated information is presented in this appendix.

1. Table A.1 lists the genes in the *50TG* collection.

2. Table A.2 lists the genes in the *115IFN* collection.

3. The genes in the *Math1*, *Mea*, and *Sey* collections are still under investigation and are not presented.

Table A.1: The 50 genes of the *50TG* data set.

Gene Symbol	Unigene ID	LocusLink ID	**Official Gene Name**
A2M	Mm.30151	232345	alpha-2-macroglobulin
ABL1	Mm.1318	11350	v-abl Abelson murine leukemia oncogene 1
APBA1	Mm.22879	108119	amyloid beta (A4) precursor protein-binding, family A, member 1
APBB1	Mm.38469	11785	amyloid beta (A4) precursor protein-binding, family B, member 1
APLP1	Mm.2381	11803	amyloid beta (A4) precursor-like protein 1
APLP2	Mm.19133	11804	amyloid beta (A4) precursor-like protein 2
APOE	Mm.305152	11816	apolipoprotein E
APP	Mm.277585	11820	amyloid beta (A4) precursor protein
ATOH1	Mm.57229	11921	atonal homolog 1 (Drosophila)
BRCA1	Mm.244975	12189	breast cancer 1
BRCA2	Mm.236256	12190	breast cancer 2
CDK5	Mm.298798	12568	cyclin-dependent kinase 5
CDK5R	Mm.142275	12569	cyclin-dependent kinase 5, regulatory subunit (p35)
CDK5R2	Mm.288703	12570	cyclin-dependent kinase 5, regulatory subunit 2 (p39)
DAB1	Mm.289682	13131	disabled homolog 1 (Drosophila)
DLL1	Mm.4875	13388	delta-like 1 (Drosophila)
DNMT1	Mm.128580	13433	DNA methyltransferase (cytosine-5) 1
EGFR	Mm.8534	13649	epidermal growth factor receptor
ERBB2	Mm.290822	13866	v-erb-b2 erythroblastic leukemia viral oncogene homolog 2
ETS1	Rn.88756	24356	E26 avian leukemia oncogene 1, 5′ domain
FOS	Mm.246513	14281	FBJ osteosarcoma oncogene
FYN	Mm.4848	14360	Fyn proto-oncogene
GLI	Mm.336839	14632	GLI-Kruppel family member GLI
GLI2	Mm.273292	14633	GLI-Kruppel family member GLI2
GLI3	Mm.5098	14634	GLI-Kruppel family member GLI3
JAG1	Mm.22398	16449	jagged 1
KIT	Mm.247073	16590	kit oncogene
LRP1	Mm.271854	16971	low density lipoprotein receptor-related protein 1
LRP8	Mm.276656	16975	low density lipoprotein receptor-related protein 8, apolipoprotein
MAPT	Mm.1287	17762	microtubule-associated protein tau
MYC	Mm.2444	17869	myelocytomatosis oncogene
NOTCH1	Mm.290610	18128	Notch gene homolog 1 (Drosophila)
NRAS	Mm.256975	18176	neuroblastoma ras oncogene
PAX2	Mm.192158	18504	paired box gene 2
PAX3	Mm.1371	18505	paired box gene 3
PSEN1	Mm.998	19164	presenilin 1
PSEN2	Mm.330850	19165	presenilin 2
PTCH	Mm.138472	19206	patched homolog
RELN	Mm.3057	19699	reelin
ROBO1	Mm.310772	19876	roundabout homolog 1 (Drosophila)
SHC1	Mm.86595	20416	src homology 2 domain-containing transforming protein C1
SHH	Mm.57202	20423	sonic hedgehog
SMO	Mm.29279	20596	smoothened homolog (Drosophila)
SRC	Mm.22845	20779	Rous sarcoma oncogene
TGFB1	Mm.248380	21803	transforming growth factor, beta 1
TRP53	Mm.222	22059	transformation related protein 53
VLDLR	Mm.4141	22359	very low density lipoprotein receptor
WNT1	Mm.1123	22408	wingless-related MMTV integration site 1
WNT2	Mm.33653	22413	wingless-related MMTV integration site 2
WNT3	Mm.5188	22415	wingless-related MMTV integration site 3

Table A.2: The 115 genes in the *115IFN* data set.

Gene Symbol	Unigene ID	LocusLink ID	**Official Gene Name**
ADAR	Mm.316628	56417	adenosine deaminase, RNA-specific
AIM1	Mm.292082	11630	absent in melanoma 1
AKAP8	Mm.328945	56399	A kinase (PRKA) anchor protein 8
ANKFY1	Mm.10313	11736	ankyrin repeat and FYVE domain containing 1
APOD	Mm.2082	11815	apolipoprotein D
AZI2	Mm.92705	27215	5-azacytidine induced gene 2
B2M	Mm.163	12010	beta-2 microglobulin
BLMH	Mm.22876	104184	bleomycin hydrolase
CASP12	Mm.42163	12364	caspase 12
CASP4	Mm.1569	12363	caspase 4, apoptosis-related cysteine peptidase
CD47	Mm.31752	16423	CD47 antigen (Rh-related antigen, integrin-associated signal transducer)
CREM	Mm.5244	12916	cAMP responsive element modulator
CXCL10	Mm.877	15945	chemokine (C-X-C motif) ligand 10
CYP26A1	Mm.42230	13082	cytochrome P450, family 26, subfamily a, polypeptide 1
DAXX	Mm.271809	13163	Fas death domain-associated protein
DDX24	Mm.3935	27225	DEAD (Asp-Glu-Ala-Asp) box polypeptide 24
DSC2	Mm.280547	13506	desmocollin 2
ETNK1	Mm.272548	75320	ethanolamine kinase 1
G1P2	Mm.4950	53606	ISG15 ubiquitin-like modifier
G7E	Mm.22506	110956	DNA segment, Chr 17, human D6S56E 5
GADD45G	Mm.281298	23882	growth arrest and DNA-damage-inducible 45 gamma
GATA6	Mm.329287	14465	GATA binding protein 6
GBP2	Mm.24038	14469	guanylate nucleotide binding protein 2
GBP4	Mm.45740	55932	guanylate nucleotide binding protein 4
GNA13	Mm.193925	14674	guanine nucleotide binding protein, alpha 13
GNB4	Mm.139192	14696	guanine nucleotide binding protein, beta 4
H2-BL	Mm.195061	14963	histocompatibility 2, blastocyst
H2-D1	Mm.374036	14964	histocompatibility 2, D region locus 1
H2-K1	Mm.379883	14972	histocompatibility 2, K1, K region
H2-L	Mm.33263	14980	histocompatibility 2, D region
H2-M3	Mm.14437	14991	histocompatibility 2, M region locus 3
H2-Q1	Mm.33263	15006	histocompatibility 2, Q region locus 1
H2-Q10	Mm.88795	15007	histocompatibility 2, Q region locus 10
H2-Q2	Mm.33263	15013	histocompatibility 2, Q region locus 2
H2-Q7	Mm.33263	15018	histocompatibility 2, Q region locus 7
H2-T10	Mm.195061	15024	histocompatibility 2, T region locus 10
HAS2	Mm.5148	15117	hyaluronan synthase 2
HTR1D	Mm.40573	15552	5-hydroxytryptamine (serotonin) receptor 1D
IFI1	Mm.29938	15944	immunity-related GTPase family, M
IFI16	Mm.227595	15951	interferon activated gene 204
IFI202B	Mm.218770	26388	interferon activated gene 202B
IFI203	Mm.261270	15950	interferon activated gene 203
IFI205	Mm.218770	226695	interferon activated gene 205
IFI35	Mm.45558	70110	interferon-induced protein 35
IFI47	Mm.24769	15953	interferon gamma inducible protein 47
IFIH1	Mm.136224	71586	interferon induced with helicase C domain 1
IFIT1	Mm.6718	15957	interferon-induced protein with tetratricopeptide repeats 1
IFIT2	Mm.2036	15958	interferon-induced protein with tetratricopeptide repeats 2
IFIT3	Mm.271850	15959	interferon-induced protein with tetratricopeptide repeats 3
IFITM3	Mm.141021	66141	interferon induced transmembrane protein 3
IGTP	Mm.33902	16145	interferon gamma induced GTPase
IIGP1	Mm.261140	60440	interferon inducible GTPase 1
IIGP2	Mm.33902	54396	interferon inducible GTPase 2
IL13RA1	Mm.24208	16164	interleukin 13 receptor, alpha 1
IL15	Mm.4392	16168	interleukin 15
IL15RA	Mm.200196	16169	interleukin 15 receptor, alpha chain
IL6	Mm.1019	16193	interleukin 6
IRF1	Mm.105218	16362	interferon regulatory factor 1
IRF5	Mm.6479	27056	interferon regulatory factor 5

Genes in *115IFN* data set (Table A.2 continued).

Gene Symbol	Unigene ID	LocusLink ID	**Official Gene Name**
IRF7	Mm.3233	54123	interferon regulatory factor 7
ISG20	Mm.322843	57444	interferon-stimulated protein
ISGF3G	Mm.2032	16391	interferon dependent positive acting transcription factor 3 gamma
JAK2	Mm.275839	16352	Janus kinase 2
JARID1A	Mm.45767	214899	jumonji, AT rich interactive domain 1A (Rbp2 like)
LGALS3BP	Mm.3152	19039	lectin, galactoside-binding, soluble, 3 binding protein
LGALS8	Mm.171186	56048	lectin, galactose binding, soluble 8
LGALS9	Mm.341434	16859	lectin, galactose binding, soluble 9
LY6E	Mm.788	17069	lymphocyte antigen 6 complex, locus E
MAFK	Mm.157313	17135	v-maf musculoaponeurotic fibrosarcoma oncogene family, protein K (avian)
MOV10	Mm.1597	17454	Moloney leukemia virus 10
MPEG1	Mm.3999	17476	macrophage expressed gene 1
MX1	Mm.33996	17857	myxovirus (influenza virus) resistance 1
MX2	Mm.14157	17858	myxovirus (influenza virus) resistance 2
MYD88	Mm.213003	17874	myeloid differentiation primary response gene 88
N4BP1	Mm.25117	80750	cDNA sequence BC004022
NDG1	Mm.26006	368204	Nur77 downstream gene 1
NDST2	Mm.4084	17423	N-deacetylase/N-sulfotransferase (heparan glucosaminyl) 2
NMI	Mm.7491	64685	N-myc (and STAT) interactor
NYREN18	Mm.5856	53312	RIKEN cDNA 6330412F12
OGFR	Mm.250418	72075	opioid growth factor receptor
PBEF1	Mm.202727	59027	pre-B-cell colony-enhancing factor 1
PELI1	Mm.28957	67245	pellino 1
PLSCR2	Mm.10306	18828	phospholipid scramblase 2
PML	Mm.278985	18854	promyelocytic leukemia
PNP	Mm.17932	18950	purine-nucleoside phosphorylase
PRKR	Mm.378990	19106	eukaryotic translation initiation factor 2-alpha kinase 2
PSMB10	Mm.787	19171	proteasome (prosome, macropain) subunit, beta type 10
PSMB8	Mm.180191	16913	proteosome (prosome, macropain) subunit, beta type 8 (large multifunctional peptidase 7)
PSMB9	Mm.390983	16912	proteosome (prosome, macropain) subunit, beta type 9 (large multifunctional peptidase 2)
PSME1	Mm.830	19186	proteasome (prosome, macropain) 28 subunit, alpha
PSME2	Mm.15793	19188	proteasome (prosome, macropain) 28 subunit, beta
PTPN13	Mm.3414	19249	protein tyrosine phosphatase, non-receptor type 13
RNPEPL1	Mm.200971	108657	arginyl aminopeptidase (aminopeptidase B)-like 1
SAMHD1	Mm.248478	56045	SAM domain and HD domain, 1
SERPINB9	Mm.272569	20723	serine (or cysteine) peptidase inhibitor, clade B, member 9
SFMBT2	Mm.329991	353282	Scm-like with four mbt domains 2
SLFN2	Mm.278689	20556	schlafen 2
SMYD2	Mm.156895	226830	SET and MYND domain containing 2
SOCS1	Mm.130	12703	suppressor of cytokine signaling 1
SOCS2	Mm.4132	216233	suppressor of cytokine signaling 2
STAT1	Mm.277406	20846	signal transducer and activator of transcription 1
STX3	Mm.272264	20908	syntaxin 3
STXBP1	Mm.278865	20910	syntaxin binding protein 1
TAP1	Mm.207996	21354	transporter 1, ATP-binding cassette, sub-family B (MDR/TAP)
TAP2	Mm.14814	21355	transporter 2, ATP-binding cassette, sub-family B (MDR/TAP)
TAPBP	Mm.154457	21356	TAP binding protein
TCIRG1	Mm.271689	27060	T-cell, immune regulator 1, ATPase, H+ transporting, lysosomal V0 protein A3
TGTP	Mm.15793	21822	T-cell specific GTPase
TRIM21	Mm.321227	20821	tripartite motif protein 21
TRIM25	Mm.248445	217069	tripartite motif protein 25
UBE1L	Mm.277125	74153	ubiquitin-activating enzyme E1-like
USP18	Mm.326911	24110	ubiquitin specific peptidase 18
VIG1	Mm.24045	58185	radical S-adenosyl methionine domain containing 2
XDH	Mm.11223	22436	xanthine dehydrogenase
ZFP36	Mm.389856	22695	zinc finger protein 36

Appendix B

NMF Statistics

This appendix contains average relative error and convergence percentage information for the five data sets.

1. Table B.1 displays the average relative error of the NMF under different constraints compared to the SVD for the 50TG data set.

2. Table B.2 displays the percentage of NMF runs that either converged or reached 1,000 iterations for the 50TG data set.

3. Table B.3 displays the average relative error of the NMF under different constraints compared to the SVD for the 50TG data set with a 0.8 global weight threshold enforced.

4. Table B.4 displays the percentage of NMF runs that either converged or reached 1,000 iterations for the 50TG data set with a 0.8 global weight threshold enforced.

5. Table B.5 displays the average relative error of the NMF under different constraints compared to the SVD for the 115IFN data set.

6. Table B.6 displays the percentage of NMF runs that either converged or reached 1,000 iterations for the 115IFN data set.

7. Table B.7 displays the average relative error of the NMF under different constraints compared to the SVD for the 115IFN data set with a 0.8 global weight threshold enforced.

8. Table B.8 displays the percentage of NMF runs that either converged or reached 1,000 iterations for the 115IFN data set with a 0.8 global weight threshold enforced.

9. Table B.9 displays the average relative error of the NMF under different constraints compared to the SVD for the Math1 data set.

10. Table B.10 displays the percentage of NMF runs that either converged or reached 1,000 iterations for the Math1 data set.

11. Table B.11 displays the average relative error of the NMF under different constraints compared to the SVD for the Math1 data set with a 0.8 global weight threshold enforced.

12. Table B.12 displays the percentage of NMF runs that either converged or reached 1,000 iterations for the Math1 data set with a 0.8 global weight threshold enforced.

13. Table B.13 displays the average relative error of the NMF under different constraints compared to the SVD for the Mea data set.

14. Table B.14 displays the percentage of NMF runs that either converged or reached 1,000 iterations for the Mea data set.

15. Table B.15 displays the average relative error of the NMF under different constraints compared to the SVD for the Mea data set with a 0.8 global weight threshold enforced.

16. Table B.16 displays the percentage of NMF runs that either converged or reached 1,000 iterations for the Mea data set with a 0.8 global weight threshold enforced.

17. Table B.17 displays the average relative error of the NMF under different constraints compared to the SVD for the Sey data set.

18. Table B.18 displays the percentage of NMF runs that either converged or reached 1,000 iterations for the Sey data set.

19. Table B.19 displays the average relative error of the NMF under different constraints compared to the SVD for the Sey data set with a 0.8 global weight threshold enforced.

20. Table B.20 displays the percentage of NMF runs that either converged or reached 1,000 iterations for the Sey data set with a 0.8 global weight threshold enforced.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	0.37%	0.38%	0.36%	0.78%	0.28%
4	0.90%	1.08%	0.90%	1.85%	0.77%
6	1.75%	1.80%	1.61%	2.69%	1.48%
8	2.10%	2.11%	1.83%	3.27%	1.44%
10	2.15%	2.65%	1.99%	3.73%	1.54%
15	4.13%	4.13%	2.95%	5.18%	2.41%
20	4.84%	4.54%	3.76%	7.34%	4.03%
25	7.80%	7.73%	5.88%	11.28%	4.99%
30	12.77%	11.89%	9.43%	18.46%	8.56%

Table B.1: Average relative error of the converging NMF runs for the *50TG* collection.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	100.00%	92.59%	100.00%	18.52%	76.30%
4	100.00%	88.89%	100.00%	10.37%	69.63%
6	100.00%	85.19%	100.00%	8.89%	51.85%
8	100.00%	81.48%	96.30%	10.37%	45.93%
10	100.00%	77.78%	96.30%	8.89%	45.19%
15	100.00%	81.48%	96.30%	8.89%	33.33%
20	100.00%	96.30%	96.30%	6.67%	26.67%
25	100.00%	88.89%	96.30%	4.44%	22.96%
30	100.00%	96.30%	96.30%	2.22%	24.44%

Table B.2: Percentage of NMF runs for the *50TG* collection that converged.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	0.05%	0.06%	0.06%	0.07%	0.05%
4	0.70%	0.45%	0.65%	0.19%	0.31%
6	0.63%	0.52%	0.65%	0.58%	0.63%
8	1.08%	0.67%	0.62%	0.87%	0.84%
10	1.27%	0.85%	0.63%	1.07%	1.05%
15	2.31%	1.64%	1.69%	1.74%	1.94%
20	2.57%	2.35%	2.16%	2.31%	2.70%
25	5.96%	3.93%	5.08%	3.48%	3.06%
30	9.06%	9.68%	9.91%	5.58%	5.61%

Table B.3: Average relative error of the converging NMF runs for the *50TG.8* collection.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	100.00%	79.17%	100.00%	15.83%	51.67%
4	100.00%	83.33%	100.00%	11.67%	27.50%
6	100.00%	83.33%	95.83%	10.00%	22.50%
8	100.00%	83.33%	95.83%	10.00%	15.00%
10	100.00%	87.50%	95.83%	8.33%	8.33%
15	100.00%	87.50%	91.67%	6.67%	5.83%
20	100.00%	91.67%	79.17%	5.00%	4.17%
25	100.00%	87.50%	75.00%	3.33%	5.83%
30	100.00%	91.67%	70.83%	3.33%	5.00%

Table B.4: Percentage of NMF runs for the *50TG.8* collection that converged.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	0.25%	0.27%	0.25%	0.75%	0.20%
4	0.76%	0.89%	0.85%	1.68%	0.71%
6	1.22%	1.33%	1.18%	2.49%	1.09%
8	1.58%	1.67%	1.47%	2.71%	1.39%
10	1.86%	2.12%	1.83%	3.64%	1.95%
15	2.61%	2.95%	2.70%	5.87%	5.14%
20	3.52%	3.70%	3.27%	7.38%	6.45%
25	4.77%	4.76%	4.39%	8.67%	7.80%
30	4.90%	5.21%	4.71%	8.99%	9.27%

Table B.5: Average relative error of the converging NMF runs for the *115IFN* collection.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	100.00%	92.59%	100.00%	25.19%	52.59%
4	100.00%	81.48%	100.00%	14.81%	30.37%
6	100.00%	85.19%	100.00%	11.85%	29.63%
8	100.00%	88.89%	96.30%	11.11%	21.48%
10	100.00%	81.48%	96.30%	10.37%	11.85%
15	100.00%	85.19%	96.30%	8.89%	4.44%
20	100.00%	88.89%	96.30%	6.67%	4.44%
25	100.00%	92.59%	96.30%	4.44%	2.22%
30	100.00%	85.19%	92.59%	4.44%	2.22%

Table B.6: Percentage of NMF runs for the *115IFN* collection that converged.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	0.02%	0.02%	0.02%	0.03%	0.02%
4	0.18%	0.22%	0.14%	0.10%	0.10%
6	0.37%	0.45%	0.41%	0.33%	0.34%
8	0.44%	0.49%	0.60%	0.41%	0.36%
10	0.79%	0.75%	0.56%	0.41%	0.41%
15	0.85%	1.28%	0.90%	0.46%	0.46%
20	1.44%	1.54%	1.41%	0.73%	0.73%
25	2.32%	2.67%	2.23%	1.50%	1.50%
30	3.94%	2.86%	2.71%	2.37%	2.22%

Table B.7: Average relative error of the converging NMF runs for the *115IFN.8* collection.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	100.00%	83.33%	100.00%	16.67%	20.00%
4	100.00%	91.67%	87.50%	10.00%	7.50%
6	100.00%	87.50%	79.17%	10.00%	5.83%
8	100.00%	91.67%	70.83%	8.33%	4.17%
10	100.00%	87.50%	70.83%	6.67%	3.33%
15	100.00%	87.50%	66.67%	6.67%	3.33%
20	100.00%	83.33%	62.50%	5.00%	1.67%
25	100.00%	91.67%	50.00%	3.33%	1.67%
30	100.00%	83.33%	50.00%	1.67%	1.67%

Table B.8: Percentage of NMF runs for the *115IFN.8* collection that converged.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	0.45%	0.48%	0.40%	0.95%	0.35%
4	1.20%	1.46%	1.05%	2.54%	0.94%
6	1.83%	1.97%	1.49%	3.59%	1.33%
8	2.40%	2.41%	1.77%	4.21%	1.64%
10	3.25%	3.33%	2.33%	5.90%	2.29%
15	5.54%	5.18%	4.09%	8.12%	4.24%
20	7.29%	6.91%	5.64%	12.35%	12.67%
25	8.04%	8.80%	8.71%	17.66%	17.88%
30	12.86%	12.94%	13.69%	27.99%	28.93%

Table B.9: Average relative error of the converging NMF runs for the *Math1* collection.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	100.00%	88.89%	100.00%	20.00%	71.11%
4	100.00%	85.19%	100.00%	11.85%	51.85%
6	100.00%	88.89%	100.00%	10.37%	34.81%
8	100.00%	92.59%	100.00%	10.37%	28.15%
10	100.00%	88.89%	96.30%	9.63%	28.15%
15	100.00%	88.89%	96.30%	8.89%	22.22%
20	100.00%	96.30%	96.30%	6.67%	8.89%
25	100.00%	92.59%	96.30%	2.22%	6.67%
30	100.00%	96.30%	96.30%	2.22%	8.89%

Table B.10: Percentage of NMF runs for the *Math1* collection that converged.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	0.05%	0.05%	0.05%	0.05%	0.05%
4	0.14%	0.22%	0.23%	0.18%	0.15%
6	0.35%	0.73%	0.58%	0.35%	0.47%
8	1.11%	0.87%	1.23%	0.31%	0.31%
10	1.01%	0.24%	0.20%	0.43%	0.37%
15	3.03%	2.45%	1.50%	0.71%	0.71%
20	1.77%	4.25%	4.06%	1.16%	1.29%
25	12.56%	13.55%	12.19%	2.11%	2.11%
30	24.42%	23.53%	22.66%	4.03%	4.03%

Table B.11: Average relative error of the converging NMF runs for the *Math1.8* collection.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	100.00%	91.67%	100.00%	20.00%	42.50%
4	100.00%	79.17%	95.83%	11.67%	13.33%
6	100.00%	79.17%	79.17%	10.00%	11.67%
8	100.00%	79.17%	83.33%	10.00%	8.33%
10	100.00%	87.50%	79.17%	9.17%	9.17%
15	100.00%	91.67%	75.00%	6.67%	6.67%
20	100.00%	95.83%	70.83%	5.00%	5.83%
25	100.00%	100.00%	75.00%	1.67%	5.00%
30	100.00%	100.00%	62.50%	1.67%	3.33%

Table B.12: Percentage of NMF runs for the *Math1.8* collection that converged.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	0.33%	0.34%	0.31%	0.51%	0.28%
4	0.97%	0.99%	0.65%	1.34%	0.64%
6	1.60%	1.31%	1.02%	1.52%	0.95%
8	1.60%	1.45%	1.03%	2.12%	0.94%
10	1.69%	1.65%	1.13%	2.34%	1.24%
15	3.30%	2.87%	2.26%	4.15%	2.36%
20	5.65%	5.04%	4.14%	7.21%	7.47%
25	9.26%	8.39%	8.04%	14.82%	15.14%
30	15.42%	15.31%	13.00%	22.28%	23.07%

Table B.13: Average relative error of the converging NMF runs for the *Mea* collection.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	100.00%	92.59%	100.00%	20.74%	55.56%
4	100.00%	92.59%	100.00%	13.33%	42.96%
6	100.00%	85.19%	100.00%	10.37%	36.30%
8	100.00%	88.89%	96.30%	9.63%	25.93%
10	100.00%	85.19%	100.00%	8.89%	22.96%
15	100.00%	85.19%	88.89%	6.67%	20.74%
20	100.00%	88.89%	96.30%	6.67%	8.89%
25	100.00%	100.00%	88.89%	2.22%	8.89%
30	100.00%	96.30%	92.59%	2.22%	6.67%

Table B.14: Percentage of NMF runs for the *Mea* collection that converged.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	0.02%	0.02%	0.02%	0.02%	0.01%
4	0.12%	0.19%	0.15%	0.12%	0.12%
6	1.20%	0.43%	0.79%	0.08%	0.07%
8	0.76%	0.55%	0.67%	0.16%	0.16%
10	0.56%	0.62%	0.42%	0.24%	0.23%
15	1.89%	1.02%	0.67%	0.36%	0.36%
20	4.37%	4.24%	4.31%	0.48%	0.53%
25	5.67%	5.51%	3.78%	0.76%	0.76%
30	11.30%	11.00%	10.35%	1.23%	1.23%

Table B.15: Average relative error of the converging NMF runs for the *Mea.8* collection.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	100.00%	70.83%	100.00%	16.67%	40.83%
4	100.00%	87.50%	87.50%	11.67%	23.33%
6	100.00%	87.50%	75.00%	10.00%	10.00%
8	100.00%	87.50%	75.00%	10.00%	8.33%
10	100.00%	87.50%	75.00%	6.67%	9.17%
15	100.00%	95.83%	75.00%	6.67%	6.67%
20	100.00%	95.83%	75.00%	5.00%	5.83%
25	100.00%	95.83%	70.83%	3.33%	5.00%
30	100.00%	91.67%	66.67%	1.67%	3.33%

Table B.16: Percentage of NMF runs for the *Mea.8* collection that converged.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	0.48%	0.51%	0.46%	1.49%	0.39%
4	1.50%	1.71%	1.53%	3.35%	1.45%
6	2.46%	2.62%	1.97%	5.52%	1.59%
8	2.98%	2.97%	2.32%	6.15%	1.47%
10	4.18%	4.12%	3.33%	8.33%	2.38%
15	6.32%	6.19%	4.73%	11.34%	4.29%
20	11.38%	11.25%	8.94%	19.51%	8.38%
25	23.23%	18.94%	18.97%	36.60%	21.77%
30	56.39%	53.81%	54.05%	N/A%	60.54%

Table B.17: Average relative error of the converging NMF runs for the *Sey* collection.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	100.00%	92.59%	100.00%	22.22%	80.00%
4	100.00%	85.19%	100.00%	13.33%	60.74%
6	100.00%	88.89%	100.00%	12.59%	49.63%
8	100.00%	92.59%	96.30%	11.85%	44.44%
10	100.00%	85.19%	96.30%	9.63%	38.52%
15	100.00%	96.30%	96.30%	6.67%	28.15%
20	100.00%	88.89%	96.30%	4.44%	27.41%
25	100.00%	96.30%	96.30%	2.22%	25.93%
30	100.00%	100.00%	96.30%	0.00%	20.00%

Table B.18: Percentage of NMF runs for the *Sey* collection that converged.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	0.05%	0.05%	0.04%	0.07%	0.04%
4	0.33%	0.46%	0.18%	0.22%	0.15%
6	1.74%	1.42%	1.19%	0.33%	0.52%
8	0.26%	0.79%	0.71%	0.42%	1.03%
10	0.54%	0.60%	1.66%	0.61%	0.74%
15	4.89%	3.73%	1.51%	1.17%	1.18%
20	9.66%	8.21%	7.17%	1.97%	1.98%
25	16.36%	19.91%	14.94%	4.04%	4.04%
30	66.87%	61.84%	62.69%	13.12%	17.45%

Table B.19: Average relative error of the converging NMF runs for the *Sey.8* collection.

k	no constraints	W smooth	H smooth	W sparse	H sparse
2	100.00%	83.33%	100.00%	18.33%	57.50%
4	100.00%	79.17%	95.83%	11.67%	35.00%
6	100.00%	91.67%	95.83%	11.67%	20.83%
8	100.00%	87.50%	91.67%	10.00%	13.33%
10	100.00%	91.67%	91.67%	8.33%	11.67%
15	100.00%	95.83%	79.17%	5.00%	8.33%
20	100.00%	95.83%	83.33%	3.33%	6.67%
25	100.00%	95.83%	70.83%	1.67%	6.67%
30	100.00%	100.00%	70.83%	1.67%	2.50%

Table B.20: Percentage of NMF runs for the *Sey.8* collection that converged.

Vita

Kevin Erich Heinrich was born in Würzburg, Germany on October 24, 1979. He graduated Valedictorian from Maryville High School in Maryville, Tennessee in 1997. He received Top Graduate Honors and a Bachelor of Science degree in Computer Science and Honors Mathematics with a minor in Economics from the University of Tennessee in May 2001 and later earned a Masters degree in Computer Science from the University of Tennessee in August 2004. Interested in continuing his current projects, he remained at the University at Tennessee to receive a Doctor of Philosophy degree in Computer Science in May 2007. He plans to explore the commercial world by starting up a company called Computable Genomix.