2016-12-01

A Framework for Evaluating Recommender Systems

Michael Gabriel Bean
Brigham Young University

A Framework for Evaluating Recommender Systems

Michael Gabriel Bean

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Arts

Department of Linguistics & English Language

Brigham Young University

ABSTRACT

A Framework for Evaluating Recommender Systems

Michael Gabriel Bean
Department of Linguistics & English Language, BYU
Master of Arts

Prior research on text collections of religious documents has demonstrated that viable recommender systems in the area are lacking, if not non-existent, for some datasets. For example, both `www.LDS.org` and `scriptures.byu.edu` are websites designed for religious use. Although they provide users with the ability to search for documents based on keywords, they do not provide the ability to discover documents based on similarity. Consequently, these systems would greatly benefit from a recommender system.

This work provides a framework for evaluating recommender systems and is flexible enough for use with either website. Such a framework would identify the best recommender system that provides users another way to explore and discover documents related to their current interests, given a starting document. The framework created for this thesis, *RelRec*, is attractive because it compares two different recommender systems. Documents are considered relevant if they are among the nearest neighbors, where "nearest" is defined by a particular system's similarity formula.

We use *RelRec* to compare output of two particular recommender systems on our selected data collection. *RelRec* shows that LDA recommeder outperforms the *TF-IDF* recommender in terms of coverage, making it preferable for LDS-based document collections.

Keywords: LDA, TF-IDF, recommendation systems, LDS Scripture Citation Index, *RelRec*, topic modeling

ACKNOWLEDGMENTS

I thank **Dr. Deryle Lonsdale** for his comprehension of computational linguistics and his ability to teach it with zeal and effectiveness. He kept me in my seat, awestruck by how amazing studying syntax could be.

I thank **Dr. Eric Ringger** for his comprehension of natural language processing and his ability to teach critically rather than by rote. He tackled every question with ease and foresight.

I thank **Dr. Alan K. Melby** for his vigor and zeal. He taught me that linguistics can be fun, even if computers are not involved, teaching me that even the humanities side of things can leave you pondering for days.

I thank **Dr. Mark Davies** for being a friend and colleague. Being able to participate in helping prepare the corpora for GloWbE (Davies, 2013), one of the largest English dialect corpora in the world, was an amazing experience to say the least!

I thank **LoriAnne Spear** for being a great department secretary, always keeping things in order. She even corrected the formatting of this very page not knowing I would acknowledge her attention to detail.

I wish to thank my wife, **Wendy Bean**, who guided me to this field of study and supported me throughout it. She has been my friend, peer, and confidant. She has proven to be true help-meet for me, especially when times get tough (such as when a database gets corrupted or when I think you are a victim of an amazingly pernicious virus and is willing to help you change all your passwords at a moment's notice—true story).

I thank my **daughter** for spilling cereal on me when I needed a reason to take a break, and for being one of my best friends. Of course, I have to thank my **son** for distracting her more often than not.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Information retrieval within large sets of textual data is a common problem. Two main approaches to this problem are search and recommendation. Search involves a user entering a text-based query and the computer retrieving documents based on that query. An example of this is the keyword-based Google search engine. A weakness with search-based approaches is that rely on a user having preconceived notions of what is within the dataset and how to determine the best search query to access the desired information using vocabulary specific to the dataset. The goal of a recommender system (RS) is to generate meaningful recommendations for items or products, either based on content similarity or—in the case of user-specific recommendations—based on content as well as user meta-data (e.g. profiles, history, ratings, and social network). Real-world operational examples of large-scale recommender systems involve suggestions for books on Amazon or movies on Netflix. "The design of such recommendation engines depends on the domain and the particular characteristics of the data available" (Melville and Sindhwani, 2010). Like query systems, recommender systems have limitations and biases based on the specific algorithms chosen, available data, and domain.

In this thesis we address the subject of recommender systems for large textual datasets, in particular with the corpus of LDS General Conference talks (*GC*). This dataset grows at least bi-annually and presently contains over 5k documents. It is frequently accessed because it is available to the public on the web. The LDS Church, consisting of over 15 million members (see `http://www.mormonnewsroom.org/facts-and-statistics`), asks its members to use *GC* as a source of personal study and a teaching resource. Therefore it is important for users to be able to locate documents that have meaningful and pertinent connections—by topic if possible. The sheer size of the corpus makes it difficult to sift through. Computational helps are already available but more are possible.

Currently users who desire to use LDS religious documents found in the *GC* can use three websites to help them: `LDS.org`, `scriptures.byu.edu`, and the BYU Corpora General Conference website[1]. The first two of these sites are similar in that they have a query system that returns entire documents from a collection. However, they do differ. At `LDS.org` users can perform a search or browse by topic. At `scriptures.byu.edu` users can search for talks by using the parameters of year, speaker, and user query. In both instances, a user may lack vocabulary needed to find desired talks/information. The BYU GC website is powerful, especially for more advanced users. It calculates word frequency, creates colorful visualizations of key-word(s) in context, and allows per-speaker granularity by utilizing its virtual corpora abilities. These operations generally start by accepting a text query which also accept part of speech specifications.

This thesis presents a new framework for evaluating different recommender systems. In doing so we compare two ways of building a recommender system. They are evaluated using the catalog coverage metric.

[1]`http://www.lds-general-conference.org/`

Chapter 2

Review of Literature: Algorithms, Models, and Evaluation

2.1 Introducing Query Search and Recommender Systems

In our information-based society consumers of text, such as researchers, want to derive useful data or make meaningful connections from textual corpora. This is a problem of information retrieval (IR) and related applications such as query search (QS), recommendation (or recommender) systems (RS), and automatic summarization (Mani, 2001). These IR applications are not without drawbacks. QS, for example, often requires that a user have at least some knowledge of the corpus being queried, including some knowledge of pertinent vocabulary in the corpus. For people who are inexperienced with a body of data (e.g. new Church members accessing the *GC*), this is not an entirely reasonable expectation. For example, upon submitting a search query, a person might be able to locate a handful of documents of interest, but finding more can quickly become an increasingly difficult process of sifting through lower and lower-ranked search results, unless the user expands the search by trying a variety of related vocabulary. Having to submit a second or third query to Google search to find desired content is an example of this problem that Google, a long-time search engine, has yet to completely solve.

When users lack the required vocabulary, or already know of one document they like, recommender systems can be helpful. Given some starting document, a recommender system can find documents that are characterized as recommended or similar. When this is done in real time, it uses an online algorithm; when pre-computed, an offline algorithm.

Query search is a type of recommender system that uses the query itself as a document to match against all other documents. The similarities between QS and RS extend to some of the algorithms, metrics, and evaluation techniques used by them. For that reason, both QS and RS will be described in this chapter.

These systems typically involve processing phases: (1) measurable features are discovered and assigned to items in the dataset and (2) an algorithm matches items of similar features. When available, personal data (e.g. ratings, profiles, or other preferences) may be subsequently leveraged to provide personalized recommendations.

While many techniques (algorithms, models, metrics, evaluative measures) have been implemented in QS and RS, we will focus on some that are typical for use on textual corpora, namely: TF-IDF in search and recommendation, Latent Dirichlet Allocation (LDA) in recommendation, k-Nearest Neighbor (k-NN), (Collapsed) Gibbs Sampling, and Variational Inference.

2.2 Relevant Algorithms

2.2.1 TF-IDF

The first major model-algorithm is TF-IDF, which is short for term frequency-inverse document frequency. *Term frequency* measures how frequent a term is throughout the document collection; *document frequency* measures how frequent the word is within a document. These two, when multiplied together, constitute a weight that denotes the overall value of a word for a given document in a document collection. TF-IDF is built to weight words according to their overall frequency and document frequency. Words that are common throughout the corpus will have an overall low value even if they are very common within a document. These are typically function words such as conjunctions, determiners, and pronouns. Words that are not common throughout the corpus—but are common within a document are weighted, or valued, high. These high-weight words constitute the core words of the document: a topical wordprint (think 'fingerprint') of a document. Documents with similar wordprints are deemed related/similar.

TF-IDF is a bag-of-words model, which means that words of a certain range (the document) are considered together, irrespective of their relative word order. For programmers, TF-IDF is a readily available model because it is well-known, easy to understand, and is included in various open-source programming toolkits such as Apache Lucene (McCandless et al., 2010), Apache Solr (Serafini, 2013; Smiley and Pugh, 2015), NaturalNode's natural (NaturalNode, 2016), and Elasticsearch (Elasticsearch, 2016). This means that for tools such as the LDS Scripture Citation Index (SCI), both the QS and RS can potentially share a code base, maintaining a simple overall code architecture. This is a desirable characteristic for time/money-constrained engineering projects, especially those that strive for simplicity of maintenance.

Spärck Jones (1972) wrote the seminal paper on TF-IDF. Her work inspired other computer scientists such as Robertson to conduct research in TF-IDF. They collaborated to create Okapi BM25 and later improved versions, a set of functions used in document retrieval that uses TF-IDF principles. Later, Robertson worked to develop the Bing search engine. Open-source tools such as Apache Lucene (McCandless et al., 2010), Luke (Bialecki, 2013), and AntConc (Anthony, 2013) result from widespread usage of TF-IDF. The LDS SCI uses Luke, and LDS.org search results are typically so similar to LDS SCI that it appears to also use TF-IDF behind-the-scenes (or something that produces the same frequency-based results).

TF-IDF stores the weight for each word in a matrix, consisting of a vector of vectors. Each sub-vector corresponds to a document. The scalars within the vector are the weighted value of the word that the dimension represents, which is any word that is not an ignored word, otherwise known as a stopword. If no stopwords are provided to the model, every word in the document collection's vocabulary would be associated with a particular dimension in every vector.

TF-IDF weighting is inspired by Zipf's law, which led to the observation that some words are used many times while the majority of words occur less often. The weight, or value, of a word is in inversely proportional to the number of occurrences, hence IDF (inverse document frequency). "The term-frequency factor was originally thought to be indicative of document topic [Luhn 1958], and the inverse document-frequency (IDF) is reasoned [Spärck Jones 1972] on the basis of Zipf's law." (Wu et al., 2008) Note, though, that not all IDF score metrics attempt to respect Zipf's law. One such example of this is the unary IDF scoring, which simply assigns a value of 1 to all IDF scores.

Building on raw frequency to compute the TF scores of the TF-IDF model, Manning et al. (2008) mention three possible methods, namely: (1) Boolean "frequencies," (2) logarithmic scale, and (3) augmented frequency. These all affect the algorithm by adjusting the built-in bias to avoid giving excess weight to terms found in longer documents. Long documents have a better chance of having key words more frequently, but more words does not always yield additional information. Similarly, the IDF portion can be calculated in a variety of ways. Variations of IDF, such as smoothing or a probabilistic approach affect the bias of how important any given word is in a document collection. All of these variants can pose a challenge for researchers as they must determine the best biases to select for a particular project. Another way to compute document similarity for search is to use word correlation factors (WCF) (Won Lee and Ng, 2007).

TF-IDF can also drive a recommender system with all the same bias adjustments mentioned previously. Each document has a corresponding TF-IDF vector. Researchers employ a variety of algorithms to compare the vectors to make meaningful connections. Where more than one textual metadatum exists for each document, each metadatum can be considered separately, then weighted appropriately (Manning et al., 2008). Examples of this would be textual titles and textual content. Although separate, they may both be used for matching purposes. Wu et al. (2008) provide a novel use of TF-IDF by creating both local relevance that "only applies to a specific document location, and [non-local], common, type is the 'document-wide' relevance that applies to the entire document. The model combines the local relevance for every location of a document by the document-wide relevance decision of the document." So TF-IDF continues to be employed.

Overall, TF-IDF is considered to be a powerful tool for information retrieval by experts in the field. Robertson (2004) said, "The class of weighting schemes known generically as TF*IDF[1], which involves multiplying the IDF measure (possibly one of a number of variants) by a TF measure (again possibly one of a number of variants, not just the raw count) have proved extraordinarily robust and difficult to beat, even by much more carefully worked out models and theories." A notable example includes Netflix's use of context (Bell and Koren, 2007).

[1]Note the asterisk in place of the hyphen.

2.2.2 LDA: A Model and Algorithms to Build It

Other researchers have continued to work on ways to improve text document collection modeling. Another text corpus technique involves Latent Dirichlet Allocation (LDA) (Blei et al., 2003a). LDA can overcome weaknesses in previous models, since TF-IDF "provides a relatively small amount of reduction in description length and reveals little in the way of inter- or intra-document statistical structure." Pre-dating LDA, the models used by researchers were Latent Semantic Indexing (LSI) and later Probabilistic Latent Semantic Indexing (pLSI). Blei et al. (2003a) describe pLSI as follows:

> "pLSI is incomplete in that it provides no probabilistic model at the level of documents. In pLSI, each document is represented as a list of numbers (the mixing proportions for topics), and there is no generative probabilistic model for these numbers. This leads to several problems: (1) The number of parameters in the model grows linearly with the size of the corpus, which leads to serious problems with overfitting, and (2) It is not clear how to assign probability to a document outside of the training set."

LDA is a generative probabilistic model, so in these respects, it is an improvement over pLSI.

In machine learning, the terms 'overfit' and 'overfitting' are used to describe a model that is calibrated too much on noise. This occurs when too many variables exist (without enough data), when the model is not trained the right way, or when the model is tested with data with which it was trained. This especially occurs with models that 'memorize' data because such models are guaranteed to always give the right answer in such cases, leading to a sense of accuracy when in fact the model is biased towards the testing data. This is akin to having a line that goes through every point of data rather than generalizing and going through the 'middle' of a set of points: the moment that previously unseen, novel data points are mapped onto the space to see where they lay on the line, the model is shown to be inaccurate; it is overfitted. LDA overcomes some of the overfitting issues inherent with LSI and pLSI models.

LDA has been used by many researchers for topic modeling and it has also been applied to word-sense disambiguation (Boyd-Graber et al., 2007), DNA research (Pritchard et al., 2000; Huelsenbeck et al., 2006; Shivashankar et al., 2011), and query search (Wei and Croft, 2006). Thus LDA, like TF-IDF, is both popular and flexible for information retrieval tasks.

LDA implements Bayesian statistics. According to Jeffreys (1973), "[Bayes' theorem] is to the theory of probability what the Pythagorean theorem is to geometry." In the LDA model, the observable variables of words are used to discover the latent (hidden) variables of topics in the corpus using Bayesian statistics by using a Dirichlet, a type of probability distribution. When applied to text, LDA identifies latent topics.

Like TF-IDF, LDA is a bag-of-words model. This means that the positions of words within a given document are interchangeable. However, for cases when a model's topic tags need to be ordered (e.g. in the same order that the words originally appeared), implementations of the algorithm do exist that maintain word order. Although LDA is a bag-of-words approach, it is probabalistic, so algorithms that infer the LDA model will treat two identical documents in the corpus differently. LDA algorithms that check for identical documents (or even nearly identical documents) apparently do not exist, although such documents could easily be identified and handled with front-end processing. This would increase complexity of the algorithm which would make it less elegant. Gibbs Sampling, an algorithm for building the LDA model, will initially tag words with random topics according to mixtures described by the model's settings. During this process documents are *not* checked for equivalency or similarity. LDA contrasts with the way TF-IDF functions, which inherently treats two equivalent documents identically; because of its probabilistic nature, even if two bags of words are very similar, LDA is less consistent in treating the bags than is TF-IDF.

In deriving the LDA model, an assumption is made to allow the formula to be simplified using Bayesian rules. The simplification, as it applies to text, is the assumption that topics do not influence each other. This is a core weakness to the basic LDA model. Nevertheless, the model does well at tagging words as belonging to particular topics. For some applications, ignoring this assumption will not be appropriate. "Although standard topic models like LDA (Blei et al., 2003b) assume that topic proportions in a document are uncorrelated, there is strong evidence that topics are dependent (Blei and Lafferty, 2007; Li and McCallum, 2006): economics and politics are more likely to co-occur than economics and cooking.” Arora et al. (2013).

Receiving as input the document collection itself, LDA also accepts other parameters such as number of topics to infer, number of inner loops of the main algorithm to run, and what topic mixture is expected. The topic mixture describes how common each topic will be. This can be a difficult proposition: telling a model to find topics with certain probabilities before knowing what topics are in the corpus. Choosing to set the topics as being equiprobable is reasonable for simple approaches. Other models can even be used to influence the initial mixture, but such approaches, of course, could be considered non-eloquent and complex—some might argue that LDA is complex enough as is.

The LDA model can be inferred (i.e. computed) by using Variational Inference, Variational Bayes, Collapsed Variational Bayes (Teh et al., 2006; Blei and Jordan, 2006; Hoffman et al., 2010), Gibbs Sampling, or Collapsed Gibbs Sampling (Porteous et al., 2008). This paper uses Collapsed Gibbs Sampling for LDA as implemented in the Mallet toolkit (McCallum, 2002), an open-source machine learning package.

2.3 Using the Models to Compute Recommendations

The TF-IDF and LDA models alone are not recommender systems. However, they *are* simplified, abstracted, structured models of unstructured data. Both lend themselves to use in recommender systems by using similarity metrics that compute distances between documents. For the LDA model, a distribution over topics is used for points of comparison; for TF-IDF, vectors of token counts constitute the compared objects. By using a distance metric appropriate for the space, recommendations can be calculated.

Since TF-IDF and LDA are different model types rather than instantiations of the same model, their models end up in different spaces. (All things being equal, such as normalization settings, many TF*IDF variants would produce different models within the same space.) Although both LDA and TF-IDF contain a second-order tensor (Turney and Pantel, 2010), or matrix, vectors from each matrix is in a different space. In TF-IDF, the space is lower-bounded by 0, having an upper bound which is collection-dependent. This space can be described approximately with the following rules, some of which are stated explicitly to contrast with those of LDA:

1. The sum over the scalars in any vector does not need to sum to 1.
2. The sum over the scalars in any vector should always be positive.
3. The sum over the scalars should generally be non-zero. (A sum of 0 indicates all the words of the document were purposefully ignored by the algorithm as stopwords, or the document was mis-retrieved/removed as sometimes happens with documents posted on the internet.)

Stopwords are typically high-frequency words that provide little meaning by themselves. Examples include words 'the', and 'it'. They are sometimes referred to as functional connectors between more specific words.

For LDA, the vectors are in the probability simplex (Blei et al., 2003b), or T-space. The space can be approximately described with the following rules:

1. All scalars in any vector are non-negative.
2. All scalars in a vector lie within the range of [0, 1].
3. The sum over these scalars equals 1 for any vector.

Since TF-IDF and LDA models are in mathematically different spaces, the similarity metrics used on them differ as well. An algorithm, k-NN, can apply to both models to locate recommendations for any document, but since their spaces are different, the distance/similarity metric within k-NN must differ. For TF-IDF's vector-frequency space, a cosine similarity metric is appropriate. In LDA's space, the Hellinger or Jensen-Shannon metrics are appropriate. Hellinger is a simplification of Jensen-Shannon which increases the speed of k-NN without any negative impact (Krstovski et al., 2013). The speed-up algorithm takes advantage of the observation that the square root portion of the distance formula is not necessary to compute relative distance. (It would be required for computing the *actual* distance between documents. For recommender systems that do not expose similarities as distances, the two metrics are functionally equivalent, with Hellinger being faster (Krstovski et al., 2013).) Similarly, a simplified form of the standard metric can also speed up TF-IDF's cosine metric by removing the square root from its computation, again with the effect that exact distances are not computed.

2.4 Evaluation

Once an RS is built and recommendations are made, it is good practice to compare the results to either a baseline, or gold standard, or to the results of another RS. This same practice applies to other IR tasks such as QS whose output is ranked results. When a model is novel, human judges may help build a gold standard. Other common evaluation methods are offline evaluations and online evaluations. We next summarize these methods.

A user study requires the researcher to create a controlled study and have willing testers. Typical problems associated with these types of studies include sample bias and the challenge of making a realistic test. Recommender systems can also be tested online by recording metrics including click-through rate (CTR), link-through rate (LTR), and cite-through rate (CiTR). These measurements can be difficult to interpret because these are not controlled feedback and there is no way to ask the user why they made the choices they did like in a user study, but some seem to favor user studies (Beel and Langer, 2015).

In the offline evaluation mode, a system is either compared to some baseline or another metric is to determining goodness of results. Offline modes are more common: 69% of research database recommender systems were evaluated offline (Beel et al., 2013b). Only 7% were evaluated online, while 21% were user studies, and the remaining 6% were not evaluated at all (Beel and Langer, 2015). "Offline evaluations typically measure the accuracy of a recommender system based on a groundtruth, but also novelty or serendipity of recommendations can be measured." (Ge et al., 2010)

Beel et al. (2013a) found that "results of offline and online evaluations often contradict each other." They propose that this is due to a variety of 'human factors' and state that "We doubt that researchers will ever be able to reliably predict whether human factors affect the predictive power of offline evaluations." This leads them to conclude that unless one is working with the assumption that the ground truth (or computer's results) is inherently better than human intuition and biases, offline testing may not be ideal. As Ge et al. (2010) discuss, “high-quality recommendations need to be fitting their intended purpose and the actors behind this purpose are the ultimate judges of the quality of recommendations.” In other words, the tests used to evaluate effectiveness ought to be based upon the ultimate goal of the project.

One way to evaluate QS or RS results is to use precision and recall measures. These are used on binary variables (either the results should or should not be listed). Precision is the fraction of results that are relevant. Recall is the fraction of relevant results that were returned. Balancing precision and recall is important. By optimizing for precision, but ignoring recall, a system may leave out results that are relevant. By optimizing for recall, it might show everything as a result, inundating the user with results. To balance these measures we compute the *F_1 score* or *f-measure*, which is the harmonic mean of the two. These metrics only work when a baseline (a.k.a. gold standard) exists: when the best results are known *a priori* for some test portion of the dataset.

Having humans create a gold standard by hand is often resource-intensive (training, time, money), especially for several result sets. The workload could be reduced by filtering out obvious bad results. Intrinsic metrics could also serve to select a baseline model from automatically-selected results. An automatically-selected baseline is not strictly a gold standard. A common method to overcome these problems is to inspect results, use them in a system, and detect which results never get clicked (i.e. the CTR, CiTR, and LTR). First results may get clicked because they are listed first, so randomly switching the first and the second could help. Another approach is to determine when users agree that the results are good. For Google, this likely reflects time spent after clicking, whether other results were ever clicked, and whether the user revised the query in order to get slightly different results.

Developing a gold standard is costly for corpora that never stop growing such as *GC*, and hence may not be an option. In such cases it is pertinent to ask whether we actually need a gold standard. Does the system need to provide all the answers or just help find some? Recall that QS and RS aim to assist the user in search and discovery. For QS, a user might expect a query to locate a result immediately while for users of RS, a journey and a series of hops while navigating between related content is much more acceptable.

Spending time honing results may not be the best way to expend resources. Developing novel tools to ease other burdens may be more effective. Google improves the QS experience by detecting when users misspell words and in some cases automatically searching with a corrected version of the user's query. Google warns users when this happens, inviting them to choose to use the query as it was typed if they want to do so. For example, for the query "brds in trees", Google responds with "Showing results for *birds in trees*...Search instead for *brds in trees*" (`https://www.google.com/search?q=cts+and+dogs&oq=cts+and+dogs&aqs=chrome..69i57j69i60l5.2445j0j7&sourceid=chrome&ie=UTF-8#safe=active&q=brds+in+trees`; emphasis added). Regardless of how we approach researching QS or RS in an area without a gold standard, we may want to build some models, inspect them, and choose which to implement initially.

Serendipity is a measure for determining the quality of results. It measures a system's ability to provide results that are surprising rather than obvious (Ge et al., 2010). It does not require results to be in a specific order, just that they are surprising or novel to the user, but still relevant. Serendipity requires a baseline to exist; it is an extrinsic metric.

When results are ordered—which is the case for some search and discovery-based methods—measures in the nDCG family can be used (Wang and McCallum, 2006). nDCG measures the similarity of the results that two systems provide. When measuring two novel systems, the one with a higher nDCG measure will be considered better. However, this is an extrinsic metric; like serendipity, it requires the use of a baseline. The gold standard does not have to be complete, but it should be statistically significant, requiring sufficient queries and query results for comparison. With few query result sets, nDCG could prefer a system that randomly appears to perform better than another system, but actually underperforms.

Another metric is catalog coverage, which gauges a system's ability to provide results throughout a data collection rather than favoring certain documents often. For systems that will be used heavily by users, this may help keep results serendipitous because greater coverage will expose the user to more content. Catalog coverage does not require any gold standard or baseline; it is an intrinsic metric. Ge et al. (2010) note "Catalog coverage can be a particularly valuable measure for systems that recommend lists of items." (2010). In this thesis, we use this metric because it is sound, can be applied to multiple runs, and does not require a baseline.

Some or all of these metrics may be important for a given user. A person inexperienced with a new data collection may prefer coverage whereas a more experienced user may prefer serendipitous (unexpected) results.

The research in these related areas is fairly fragmented and disparate. In this thesis we propose is a unified framework to effectively apply recommender technology to an arbitrary document collection. To accomplish this we use a combination of open source tools and custom code to develop an evaluation framework for recommender systems.

Chapter 3

Methodology

3.1 Introduction

In the previous chapter we discussed algorithms for two recommender systems: TF-IDF and LDA. Based on the literature, our hypothesis is that the LDA-based RS would have higher catalog coverage than the TF-IDF system. Furthermore, we wanted to quantify its advantages over TF-IDF. To do that in a principled way, I designed and implemented a framework that enabled this experiment, but also provided a framework for others to carry out similar experimentation with arbitrary recommenders. The goal was to make the framework generalizable to other data collections and other recommenders.

In this chapter we introduce *RelRec*. See Figure 3.1. This chapter describes the components of the system, the output at each stage, and the processes involved in using the framework. We also discuss the associated data corpus, models, model parameters, metrics, code management, tools, and evaluation. We also present an experiment carried out with the framework. Based on this experiment, we will confirm that the LDA-based model indeed has greater catalog coverage than an TF-IDF RS that uses an off-the-shelf formula, for 1-100 recommendations. Runs consisting of more than 100 recommendations were not evaluated.

This chapter discusses each of the components of this framework:

1. corpus collection and pre-processing,

2. implementation of an LDA-based recommender system,

3. implementation of a TF-IDF recommender system, and

4. comparative evaluation of the outputs from each system.

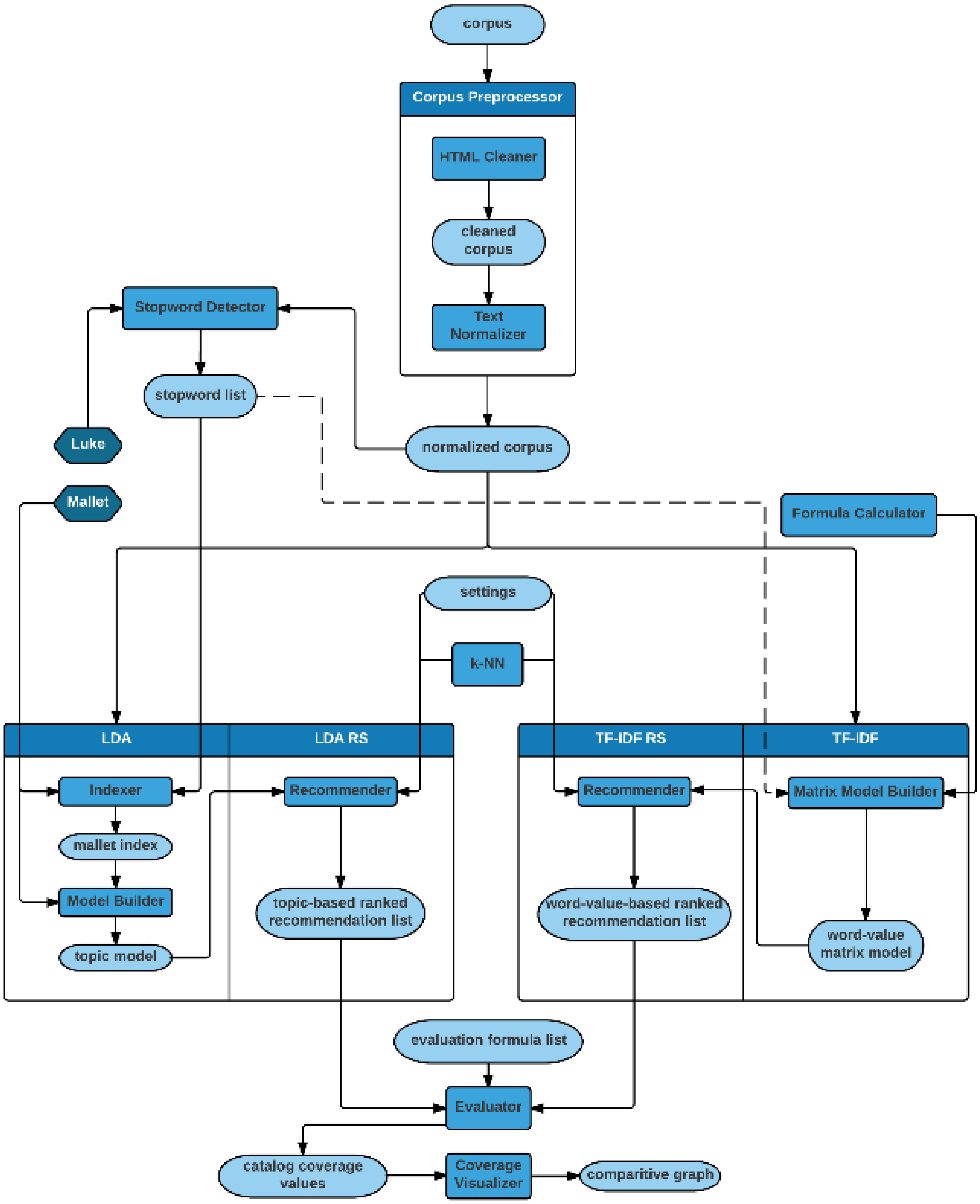

Figure 3.1: RelRec 1.0 Framework Flow. Dotted lines are optional. Hexagons are open source tools.

3.2 Corpus Characteristics and Preparation

The framework requires as input a corpus. The corpus I chose to use was the *GC* provided by Dr. Liddle as a MySQL database, which he used for LDS SCI at that time. The authorship dates for these talks fall within the range of 1942-2013. They total over 5000 documents. Some were extemporaneously given while others were scripted—sometimes being revised when speakers deviated from pre-written talks. The intended audience was either the male members of the church, female members of the church, or the entire church. The size and the extent of internationality of the audience has been increasing over time, with translations being provided in several languages. The corpus we use only includes the English versions.

3.3 Corpus Pre-Processing: Cleaning + Normalization

Initially the corpus was formatted as HTML web pages, which does not lend itself to text mining with the models under discussion. The HTML contained JavaScript, HTML elements, website boilerplate (i.e. CSS, menu, headers), titles, speaker name, images, and references (e.g. 'See 1 Cor. 15:1'). Removing all of these would simplify the content down to the essence of the corpus. TF-IDF would probably end up considering HTML elements as akin to function words and therefore automatically ignore them, but they risked confounding the LDA model. The task of converting HTML into simple text seemed at first a fairly straightforward process, but it turned out to be somewhat complicated, because the documents had been produced by a variety of different means over the years, using a number of different formats. Furthermore, JavaScript does not support as many advanced regular expression operations (e.g. positive look-behind operation) as other languages. This required more coding effort and less elaborate regular expressions. (On the flip side, less elaborate regular expressions are more readable by software developers.) After applying the regular expressions, I found that portions of some documents were due to improper HTML. After downloading more recent versions of these HTML documents, I found that by using a different set of regular expressions they could also be pared down to their core content. The end result of all this was a cleaned, normalized, text-only format of the corpus with documents produced in the date range of 1942-2013.

I wanted to ensure that the LDA and TF-IDF systems were placed on even grounds. Although tools like the Mallet toolkit would downcase input by default, I made sure to do it myself. I also removed formatting such as bold, italics, underlines, blockquotes, subscripts, and superscripts.

I removed apostrophes in contractions such as *we'll* so that contractions would be treated as readable words rather than parsed as separate words during Mallet toolbox indexing. In other words, I did not want *we'll* to be seen as *we* and *ll* in the LDA model while in the TF-IDF model it would could end up being seen as *we'll*. I could have opted to replace every instance of *'ll* with *will*, but to be more true to how the text was spoken, I opted to simply remove the apostrophes. I could have just as easily (1) left them alone with the result of having *ll* be part of the vocabulary, or (2) expanded the contractions. Either of these options was possible because I never planned to display these representations of the documents to end-users as original text. The normalized version of each documents is its simplified form for use by the recommender systems.

While apostrophes required special handling because of contractions, other punctuation was more straightforward: they were removed. This included removing punctuation including hash mark, question mark, exclamation sign, dollar sign, percentage sign, ampersand, braces, arithmetic symbols, tilde, colon, semi-colon, underscore, quote signs, comma, and full stop. In other words, anything that was not alphanumeric was removed.

Previous work showed that certain words confound the LDA model (Bean and Ringger, 2013). Although these words are important in a Christian religion, they are so common within the corpus that they confound the models in the same way function words do. The words were *jesus, christ, god,* and *gods*. These words were found by using Luke, a tool used to perform queries on a TF-IDF index. Applying lessons learned from that work meant adding these words to the default stopword list that the Mallet toolkit provides. I used the downcased forms of the words since I had downcased the corpus as mentioned earlier. The total number of stopwords was 529.

Following is a sampling of Mallet toolkit stopwords, from *a* through *com*, then skipping to *rather* through *s*:

a, able, about, above, according, accordingly, across, actually, after, afterwards, again, against, all, allow, allows, almost, alone, along, already, also, although, always, am, among, amongst, an, and, another, any, anybody, anyhow, anyone, anything, anyway, anyways, anywhere, apart, appear, appreciate, appropriate, are, around, as, aside, ask, asking, associated, at, available, away, awfully, b, be, became, because, become, becomes, becoming, been, before, beforehand, behind, being, believe, below, beside, besides, best, better, between, beyond, both, brief, but, by, c, came, can, cannot, cant, cause, causes, certain, certainly, changes, clearly, co, com . . . rather, rd, re, really, reasonably, regarding, regardless, regards, relatively, respectively, right, s . . .

Note that *cant* is a stopword (*can't* without apostrophe) but *re* is also a stopword, which is the contracted form of *are*. This list also contains *we'll* and *we're*, but without apostrophes. The default stopword list appears to have been created to fit various normalizing strategies, which means not all 529 words will apply to the corpus I use here.

3.4 Models + Parameters

This section briefly describes how the models are built and how they are used to produce recommendations. Note that the framework is flexible enough that either or both of the recommender systems could be supplanted or modified.

3.4.1 LDA RS

The LDA RS is built by using the Mallet toolkit and custom code. The Mallet toolkit is needed to build a Mallet-compatible index of the corpus. Both the cleaned corpus and the customized stopword list are parameters to the indexing process. Mallet then uses this index to generate a topic model. That model is consumed by custom code to generate a topic-based ranked recommendation list based on nearest neighbors is created. The Hellinger distance metric was used because LDA's vectors are in the probability simplex.

While other forms of LDA, such as hierarchical LDA, would be interesting to use on this corpus, it was not immediately apparent how it could be bootstrapped for such a use, so I chose instead the LDA model introduced by Blei et al. (2003a). Furthermore, I had experience using the latter.

3.4.2 TF-IDF RS

While open-source tools for TF-IDF exist, they were not readily available in nodeJS at the onset of this project. As a result, I wrote the TF-IDF code myself in nodeJS, the language of choice given its flexibility and parallelization features: I wanted to ensure that the TF-IDF code would allow for easy substitution of distance metrics for future work. The resulting code would also prove forward-compatible with later work.

The matrix model builder uses a TF-IDF formula to produce a word-value matrix, given the cleaned corpus. Optionally, stopwords may be passed to this process, but I opted not to include them because TF-IDF tends to be robust against high-frequency words. Like the LDA model, a recommender consumes the model builder's output to rank documents by similarity by using k-NN and an appropriate similarity metric—in this case, the cosine metric.

3.5 Evaluator

The evaluator takes ranked lists from the two recommender systems and does computations to quantify the results.

RelRec compared the output of both systems using the catalog coverage metric described by Ge et al. (2010). Although other metrics such as similarity or serendipity would generally be viable for comparing recommender systems (Ge et al., 2010), they are not possible here because no shared baseline or gold standard exists to which the two systems may be compared.

Following is a high-level view of the process to build and compare the systems, the last step of which is the actual comparison step:

1. Run the TF-IDF matrix builder,

2. Run k-NN with a cosine distance metric for the TF-IDF model,

3. Run Collapsed Gibbs Sampling, using Mallet, to infer the parameters of an LDA model,

4. Run k-NN with a Hellinger distance metric for the LDA model, and

5. Compute the catalog coverage metrics for each recommender.

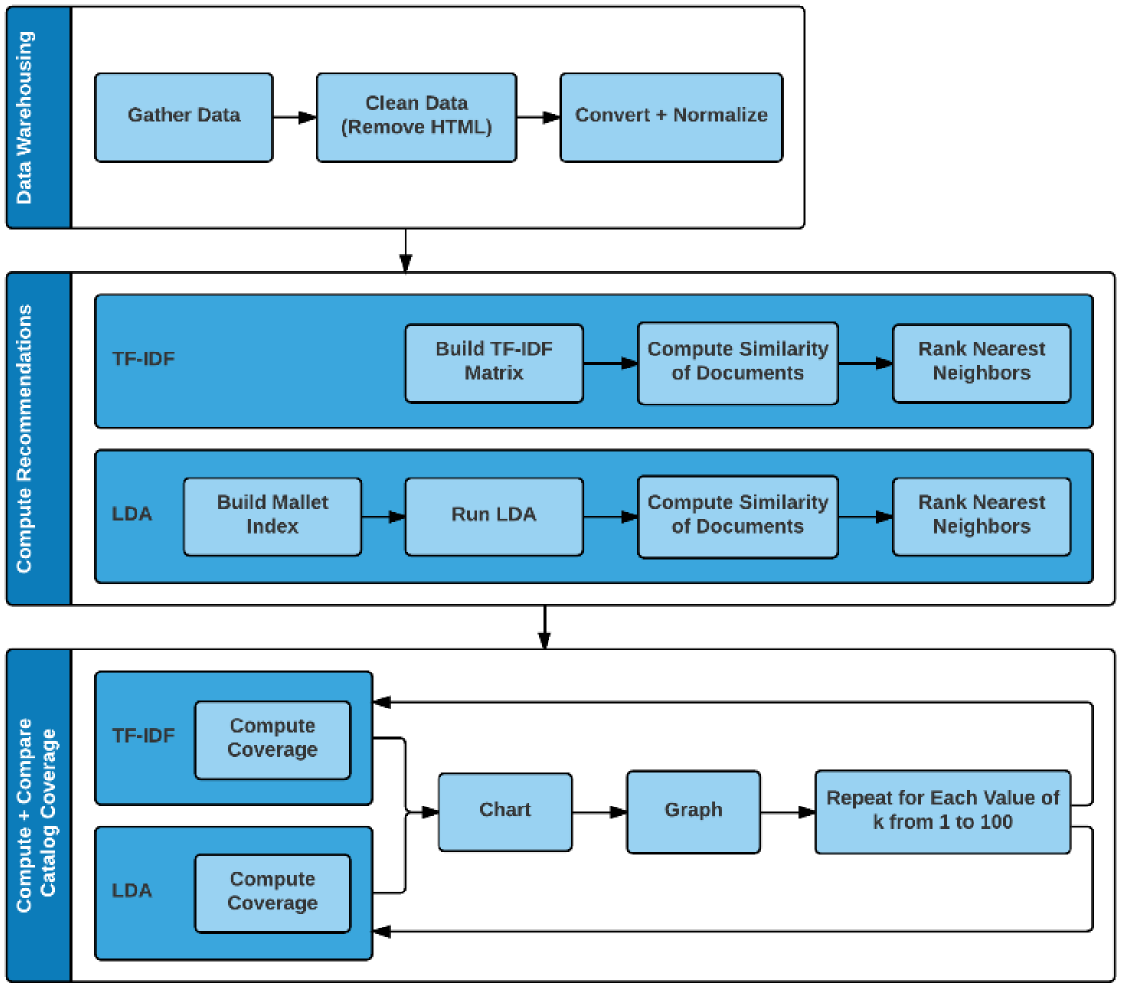

Figure 3.2: The entire process for the thesis work outlined as phases with modules. Phases that are higher on the diagram are performed before lower ones. Parallelization of some sub-phases is possible, but was not performed in this work.

Figure 3.2 illustrates the key steps and the order in which they were performed. This chapter describes each of these in detail. While some steps are not novel themselves, the way that they are pieced together is novel: the comparison of two recommender systems for the *GC* is novel for this thesis. Some code produced in particular steps was also innovative and is indicated where appropriate.

Section 3.6 discusses some of the finer details of the project. For reproducibility, Table 3.1 shows the values, settings, and configurations I selected for the algorithms.

3.6 Programming Languages

This section describes code management, programming language selection, sanity checking (helpful when debugging), and code versioning.

I used both cutting-edge programming languages and well-established ones for this project. The result is code that is easy to follow, maintainable, mainstream, and capable. The goal was to make the code configurable, re-executable, measurable, and share-able.

In software development, *cutting-edge* technologies are considered stable and new while *bleeding-edge* technology is less stable. At first I used nodeJS while it was still bleeding-edge, which proved problematic. Fortunately, nodeJS over time evolved and became more mainstream. It matured rapidly and became reliable enough to use in the final code of this project.

Many scripting and programming languages were used in this project, including bash, shell, Make, docker.io (hereafter 'docker'), docker-compose, nodeJS, Java, and YAML. MySQL was used for some database storage, and HTML was the original format for corpus documents.

I committed both the code and the corpus documents to a publicly accessible (Git) repository[1]. This tracks code modifications to the original corpus when they should not happen—or when some other process does so (e.g. Dropbox syncing).

3.7 Code Management/Organization

Early on in the project, it was clear that there would be great value in coding the main steps of the project as separate modules. Originally, this meant putting code specific to each high-level step of the project into separate folders—one for 'remove HTML', one for 'run LDA', etc.. Each folder had a main high-level function that would be individually called

[1]http://linguistics.byu.edu/thesisdata/relrec.html

Algorithm	*Variable Name*	*Variable Type*	*Input Value*	*Use*	*Reason*
k-NN	k	integer	1 through 100	Sets number of neighbors to return for each document.	Allows for commensurate comparison of systems.
Gibbs Sampling	t	integer	200	Sets number of topics to find	Previous research determined that 200 was an appropriate setting.
Gibbs Sampling	iterations	integer	1000	Sets number of inner loops of sampling to perform.	A setting that is too low will yield an incomplete model. Previous work determined 1000 iterations was sufficient.
k-NN distance metric for TF-IDF model	distance	function	cosine	Distance metric allows algorithm to measure distance between documents in the model.	Intuitive and well known. Others functions exist for this, but are left to future work to use.
k-NN distance metric for LDA model	distance	function	Hellinger	Distance metric allows algorithm to measure distance between documents in the model.	Well known and works for vectors in the probability simplex.
TF-IDF	stop words	list of word strings	none	Sets words to ignore in model.	TF-IDF is robust against high-frequency words since they automatically receive low TF and low IDF values.
Build Mallet index	stop words	list of word strings	default Mallet list + 5 hand-selected words	Remove stop words at indexing time.	Helps LDA (it is not robust against high-frequency words).
normalization	‘to lower case’	boolean	true	If true, tells algorithm to downcase all text.	Force all subsequent algorithms to ignore case by downcasing everything.

Table 3.1: Settings and parameters.

by a Makefile target. This was clean and organized, though some facets of the project were problematic at times. This was especially true when different programming language versions were required by different modules, or when I needed to upgrade to a later version of a programming language for a particular module. For example, I had to install different nodeJS versions depending on which step I was running or depending on which open source projects I included. To overcome these issues, I applied a dockerization strategy to the project, associating Makefile targets with docker containers. With docker, the code requirement surface of this project decreased, while maintainability and share-ability increased; docker abstracted away dependency management into separate docker files.

3.7.1 Docker + Docker-compose

To take full advantage of docker, I used docker-compose. This is a docker utility that makes it easy to define multiple docker containers, mount filesystem volumes to them, and set environment variables.

Docker-compose YAML files have a notion of targets, where a target corresponds to a container that can be run. They are like individual virtual machines (VMs) that can be configured and spun up on demand–each with their own environment variables, ports, volumes, and hosts file. Like the Makefile targets, each container corresponds with a high-level process of this project, such as 'build LDA model'. I build the Make code such that the docker daemon built each target upon being directed by Make to do so, and directories were mapped to the image as volumes. This allowed the docker containers to be both modular and systematic. Each docker target had a corresponding file called Dockerfile which instructed the docker daemon to obtain requisite packages or programming languages for the module that runs the code best tailored to the task of that module. For example, I used a Java-based image to run the Collapsed Gibbs Sampler because the sampler was part of the Java-based Mallet toolkit. For TF-IDF, I used a nodeJS-based image. Whenever a module's code ran and completed, the docker container terminated and the next would be allowed to be initiated by Make according to the order I defined.

Each key step of the project is coded such that a docker container exists with inputs from a previous step and with outputs to subsequent docker containers (e.g. the *clean+normalize container's* output is mapped as input to the *build TF-IDF model* container). I mapped docker inputs as read-only and outputs as read-write. This means that each module cannot modify the previous module's output. This immutability is helpful when programming because it means that although two recommender systems depend on the same corpus, they will not influence each other while each runs and builds. This is also helpful when debugging. Even simple operations on a corpus of this size can take a long time to run, especially if the programming language runs slowly when dealing with string operations, such as regular expression operations (e.g. find, find+replace). This keeps the modules from influencing each other unless directed to do so.

For simplicity, I mapped inputs and outputs to the root of each docker container's file system as */input* and */output*, respectively. The first docker container has one input: the MySQL database containing the raw corpus data, containing pre-downloaded HTML documents critical to jump-starting this project. Modularizing the steps as separate docker containers with read-only protections is apparently a novel approach to building and comparing recommender systems and an innovation introduced by this thesis.

Nomenclature of Docker-compose Targets

I named each docker target logically so that upon querying the docker daemon for the list of 'images' (i.e. list of programs); each would be intuitive to execute manually:

1. *compute_clean_and_normalize*: to normalize documents by removing boilerplate HTML, JavaScript, references from talks (nearly anything in parenthesis), then downcasing everything;
2. *build_mallet*: to compile the Mallet toolkit from source;
3. *compute_mallet_index*: to index data for use with Mallet;
4. *compute_lda*: to build the LDA model via Collapsed Gibbs Sampling
5. *compute_lda_recommendations*: to compute recommendations based on the (LDA) topic model; and

6. *compute_tf_idf_recommendations*: to build the TF-IDF matrix, then compute recommendations using it;

7. *compute_compare_recommendation_sets*: to apply the evaluative metric (catalog coverage) on results from both recommender systems such that the comparison is system-agnostic; recommendations—not recommender systems—are provided as input to this module.

3.8 Sanity Checks

Even when code is provided in toolkits such as Mallet, one must verify that the code is performing correctly. Debugging is one way to do this. Building tests is another way. Throughout the phases of this project, I tested core functions to ensure that the "numbers added up". For example, I inspected the output of the LDA model. Inspection of that model was important to know that sufficient burn-in time (iterations of Collapsed Gibbs Sampling) was used. I provide a sample of this output here.

The results of LDA as a topical index on this corpus are available at `http://bean5.github.io/lds-talks-by-topic/` which shows an interactive index of an LDA model with 150 topics.

Following is a selection of topic clusters and their most common words which LDA identified. I combine singular and plurals where appropriate. The words in the topic remain downcased since that is how the model sees the words. Although LDA does not attempt to label a topic, I label them here for reference:

1. *addressing remarks*: sisters, brothers, love, church, conference, brethren, lives, lord, people, great, work, spirit, good, grateful, bless, today, hearts, wonderful, gospel

2. *testimony*: testimony, witness, bear, true, truth, gospel, testimonies, prophet, father, lord, son, lives, joseph, smith, savior, knowledge, living, testify, power

3. *family + family responsibilities*: children, home, parents, homes, family, mother, father, teach, love, fathers, mothers, child, responsibility, son, taught, families, teaching, respect, care

4. *questions to life*: question(s), answer, spirit, heart, asked, words, mind, man, voice, soul, life, answered, heard, experience, speak, told, feeling, found

5. *lost sheep*: sheep, lost, shepherd, feed, son, father, flock, lambs, fold, back, brother, peter, parable, savior, saith, bring, shepherds, find, return

6. *early church history*: joseph, oliver, smith, cowdery, plates, church, book, witnesses, mormon, work, sacred, translation, hyrum, angel, martin, whitmer, harris, received, gold

7. *parable of the 10 virgins*: individual, oil, officers, make, soul, ideals, virgins, lamps, foolish, duty, responsibility, fellow, inspiration, truth, meet, guide, bridegroom, workers, wise

8. *virtues*: virtue, kindness, holy, dominion, righteousness, knowledge, love, presence, thoughts, meekness, confidence, faith, pure, long, gentleness, patience, spirit, persuasion, soul

9. *sabbath day*: day, sabbath, holy, sunday, thy, lord, worship, days, rest, commandment, observance, offer, seventh, unspotted, sacraments, prayer, house, fully, week

10. *fasts + offering*s: fast, fasting, day, offering, poor, prayer, offerings, law, meals, house, esther, month, principle, days, spring, time, generous, food, deal

Providing output similar to this from the TF-IDF model is not possible. The TF-IDF model, although simple and straightforward to build and understand, is basically just a series of numbers that indicate which words appear in a document and how important they are to the document and overall corpus. I did verify that they added up to 1 where they should—and they did.

3.9 Conclusion

This project incorporates many algorithms, metrics, and technologies to achieve its purpose. I paid careful attention to organization, structure, and open-source projects to build maintainable, functional, scalable, modern code.

To run this code on a corpus of similar size would theoretically require around one hour of configuration and four hours of actual run-time. This timing assumes that the source corpus has already been filtered and normalized, since that step is highly corpus-dependent. I versioned code using Git, so getting up and running with a direct copy of this project would be a matter of forking the source code, installing Make, docker, and docker-compose, then by running the Make target ‘thesis’. Docker abstracts away the remaining dependencies without installing them on the user’s system, keeping the host system clean.

Chapter 4

Results & Evaluation

The purpose of this thesis is to develop a framework for comparing two or more recommender systems in order to determine which would be best suited for an arbitrary document collection (e.g. the *GC*). In this case, one recommender leverages a TF-IDF model while the other uses an LDA (word topic) model. The models are therefore structured abstractions of the otherwise 'unstructured' text. The hypothesis was that the framework could identify a best recommender system, based on the selected evaluation metrics.

The core function of a recommender system is to independently retrieve relevant and helpful information for the user. The recommender should be able to provide users with what they desire and inspire confidence so that users are satisfied that the system has provided useful results. There are a number of characteristics upon which recommender systems can be quantitatively evaluated and compared, including accuracy (via precision, recall, or a combination of the two), serendipity, and catalog coverage. Of these, only catalog coverage is appropriate for use on this work because (1) it does not require a baseline, which *GC* does not have and (2) it does not require human judges.

Catalog coverage is a measurement of the percentage of documents that are recommended at least once in a corpus. Each catalog coverage value exists the interval of $(0, 1]$, where a 1 indicates complete coverage. (The zero-value coverage is excluded since an RS that never makes a suggestion is not really a system worth comparing.) A theoretical recommender that selects any document at random as a recommendation could achieve high coverage quickly, but would not be useful and users would likely not perceive it as intelligent (digital advertisements sometimes elicit this response). A balance must be struck between low coverage and high coverage. Because catalog coverage is one of the evaluative metrics most appropriate to this thesis, it is the main value used in charts and graphs in this chapter.

LDA uses topic mixtures to represent documents whereas TF-IDF uses term counts. LDA is a more descriptive model of underlying structure since it was engineered to locate latent topics. One goal of this thesis was to show that an LDA-based RS would have higher catalog coverage than the RS based on TF-IDF. Position (rank) of recommendation does not matter to this evaluative metric, although the system ranks recommendations from best to worst.

RelRec's process of measuring catalog coverage proceeds as follows:

1. choose k: number of top recommendations to select;
2. calculate N, the number of documents in the corpus;
3. calculate n, the number of documents recommended at least one time within k recommendations; and
4. solve for catalog coverage, C, where $C = n/N$.

Theoretically, the number of recommendations could vary from document to document, but consideration should be taken to ensure a fair comparison to another model. At a minimum, both systems should have the same number of total recommendations. Even more parity could be achieved by generating the same number of recommendations for the same document. In this work, I assume that if either system were used for LDS SCI, a fixed number of recommendations would be presented to users, regardless of document.

A sample topic cluster from the LDA model, after unsupervised learning, is *wisdom, tobacco, word, health, liquor, drink, alcohol, smoking, body, drinking, cigarette, habit, drugs, coffee, alcoholic, treasures, smoke, promise, great.* In one run of LDA, the document which is most made up of this topic is the talk by Theodore M. Burton entitled *The Word of Wisdom.* A recommendation set based solely on this topic dimension is shown in Table 4.1. This is a simplification of the real model because the LDA-based RS used by *RelRec* takes all topics into account.

One expects that if the user requests more recommendations, the catalog coverage would increase. Presenting too many recommendations initially would likely overwhelm the user by reducing usability and perception of quality. To get a better perspective on how the

Rank	Title	LDS SCI Document ID	% Similar
1	Why Be Foolish?	555	0.30396
2	Eat Flesh Sparingly	331	0.28482
3	Scientific Proof for the Word of Wisdom	212	0.27833
4	Liquor Advertising	300	0.27401
5	The Evils of Cigarette Smoking	1403	0.23218

Table 4.1: Recommendation set of size 5 for the document *The Word of Wisdom*

two recommender systems perform for various numbers of recommendations, I calculated catalog coverage for sets of recommendation of varying size. In other words, I calculate C for every value of k from 1 to 100, inclusive.

Figure 4.1 shows the catalog coverage for each recommender system, referred to by their core algorithm, either TF-IDF or LDA. Based on the values, I would suggest using a default of 5-25 recommendations, since beyond that there are diminishing returns. On average, the system that provides greater catalog coverage will expose a user to more recommendations. This may mean that the system is more serendipitous or insightful, or at least appears to be so. Proving this would require either measurements of human reaction (e.g. asking users whether the results they received were surprising yet relevant) or a baseline for comparison. Consequently, this must be left for future work.

Figure 4.1 shows only a fitted line. For more precise values (to 5 significant figures), see Table 4.2, that shows the actual measures for 1-20 recommendations. Appendix A contains tables which show all values used for evaluation in this work.

4.1 Interpreting Results

The LDA-based recommender system consistently produces recommendations that encompass more documents on average. Although the catalog coverage values are consistently higher for the LDA-based RS, this is an empirical value. As the corpus grows (or is replaced with another one entirely), LDA may not continue to prevail. However, in general LDA will perform best when many topics are inferred by the Collapsed Gibbs Sampling algorithm because at lower numbers of topics the 'fingerprint' of any given document probably

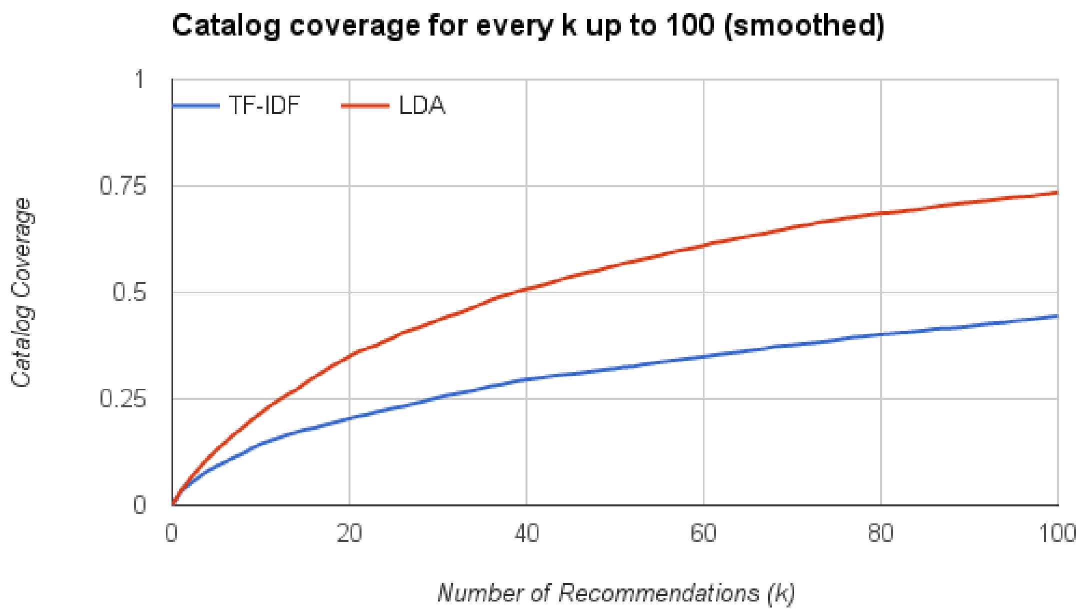

Figure 4.1: Catalog coverage values for all models (k=1 through k=100), depicted as a fitted line.

becomes less clear. For this reason, there is a solid benefit to using LDA since it yields higher catalog coverage with fewer recommendations, but not so many that it functions like a random recommender.

As the number of recommendations increases for a given document, coverage increases monotonically. This is not an empirical quality for this dataset—it will occur on any corpus because recommendations are not made more than once in any recommendation set. In other words, no list of recommendations for any document has repeats. They are mathematical sets—not lists. So as the number of recommendations increases, so does the coverage. The characteristics we need to focus on when measuring catalog coverage are rate of increase, and when its performance exceeds that of an alternate system. For this corpus, whenever 1-100 recommendations are generated, the LDA-based RS *always* has a higher catalog coverage value, achieving 13.0% coverage at 5 recommendations and 21.7% at 10 recommendations. These are both approximately 50% higher than the catalog coverage that the TF-IDF RS produces.

	Algorithm	
# of Recommendations	TF-IDF	LDA
1	0.03378	**0.03558**
2	0.05097	**0.06216**
3	0.06596	**0.08655**
4	0.08095	**0.11033**
5	0.09194	**0.13032**
6	0.10194	**0.14831**
7	0.11353	**0.16730**
8	0.12233	**0.18329**
9	0.13412	**0.20108**
10	0.14411	**0.21707**
11	0.15151	**0.23306**
12	0.15831	**0.24725**
13	0.16570	**0.26004**
14	0.17170	**0.27124**
15	0.17809	**0.28723**
16	0.18149	**0.30062**
17	0.18769	**0.31361**
18	0.19268	**0.32660**
19	0.19808	**0.33780**
20	0.20388	**0.34999**

Table 4.2: Catalog coverage values for all models (k=1 through k=20). Precision is set to 5 significant digits. Higher values for each run are shown in bold.

Chapter 5

Conclusion

5.1 Discussion

The purpose of this thesis was primarily to produce a framework for vetting recommender systems against a document collection. We illustrated this by providing the *GC* corpus as input to the framework, using TF-IDF and LDA-based recommender systems.

Such a system could be used in any application, such as `LDS.org`, that displays documents from the same corpus. In this thesis we built two systems and compared them using the catalog coverage metric. *RelRec* confirmed the hypothesis that the LDA-based RS, *RelRec*, would out-perform a TF-IDF RS.

With the needs of LDS SCI in mind, the goal was to identify which recommender system would yield a higher catalog coverage, but have intuitive results as well. Both TF-IDF and LDA were verified as providing intuitive recommendations (per visual inspection and wide-spread use), but *RelRec* showed that LDA provides the best catalog coverage when showing the 5 recommendations for each document. This is in fact true for any number of recommendations from 1 to 100, as shown in the tables and graph in Chapter 4.

In its current form, this work is ready for use with other corpora and other recommender systems. However there are simplifications, optimizations, and modifications that can be made to the base code.

5.2 Future Work

The experiment and code described in this work could be pursued in various research directions. It can be easily adapted for use with different corpora. Stopword lists can be generated dynamically. The normalization functions, distance metrics, evaluative metrics, and number of LDA-discovered topics can be easily modified.

An area where this work might be particularly interesting and novel would be to perform the same steps done in this work, but using chapters of scripture as the corpus documents. This can be applied to a restricted set of documents within a religion or to documents of multiple religions, such as The Holy Bible and The Qur'an, or The Book of Mormon and The Doctrine and Covenants. Perhaps the most interesting combination of documents for the LDS Church would be to run *RelRec* on a combined set of The Holy Bible, The Book of Mormon, The Doctrine and Covenants, and The Pearl of Great Price as input. The models could even be used to enhance current LDS scriptural footnotes. In such a case, cross-references could potentially be used as a baseline, thus enabling a larger set of available evaluative metrics than the one available to this work. The cross references in the standard works are a much smaller set than what the experts originally created, so research that consumes cross references should considering using the larger set.

Another way to build on this work would be to explore using word correlation factors (WCF). WCFs can be used to measure similarity of documents by measuring similarity of phrases. Won Lee and Ng "consider the correlation factors of different words in any two phrases of two different documents to determine the degree of similarity of the phrases, which in turns can determine the similarity of the documents based on the number of matched phrases/sentences in the documents." They also say that "experimental results show that [their] phrase-matching approach is accurate and outperforms the word-based similarity matching approach." (Won Lee and Ng, 2007) Note that they introduce WCF as a way to remedy the problem of duplicate or near-duplicate documents in search results. Similarly, depending on the application, documents that are duplicated in recommendations (e.g. a talk given a second time) can sometimes be considered to be bad recommendations. Thus a balance should be struck so that results are serendipitous yet relevant. An interpolated model, (e.g. WCF combined with TF-IDF) may produce relevant recommendations.

5.2.1 Evaluating With Other Metrics

Initially, I hoped to use the formulas as provided and described by Ge et al. (2010), but serendipity was not suitable to use because no baseline exists for *GC*. Future work can do this by comparing to a baseline; recommendations made by *RelRec* recommenders may suffice. Once a gold standard is created, even if it only exists for a subset of the documents, work may proceed with more metrics than were suitable for this work as long as the set is statically significant.

5.2.2 Other Topic Models

Other forms of LDA could be interesting to use on this dataset, such as hierarchical LDA. However, there are non-LDA topic models that are more modern. Research as early as 2012 indicates that some algorithms for inferring topic models can have provable guarantees (Arora et al., 2012), which is an improvement over LDA. One algorithm uses anchor words and assumes the topics are correlated (Arora et al., 2012). In contrast, Anandkumar et al. (2012) present an algorithm that assumes topics are not correlated. To quote Arora et al. (2012), "Both algorithms run in polynomial time, but the bounds that have been proven on their sample complexity are weak and their empirical runtime performance is slow. It is also unclear how they perform if the data does not satisfy the modeling assumptions" (Arora et al., 2013). In this same article, Arora et al. present an algorithm with provable guarantees that is also practical (it does not violate model assumptions).

> "Our algorithm performs as well as collapsed Gibbs sampling on a variety of metrics, and runs at least *an order of magnitude faster*, and as much as fifty times faster on large datasets, allowing real-time analysis of large data streams. Our algorithm inherits the *provable guarantees* of [previous algorithm] and results in simple, *practical implementations*. We view this work as combining the best of two approaches to machine learning: the tractability of statistical recovery with the robustness of maximum likelihood estimation....
>
> [These new] algorithms for topic modeling...are efficient and simple to implement yet maintain provable guarantees. " (Arora et al., 2013; emphasis added)

Their algorithms also tend to be faster. Whether the algorithms introduced by Arora et al. lend themselves to recommender systems is not mentioned in their article, but it seems intuitive that it would be the case since LDA is a topic model and was adapted for use as a recommender system for this work. Note that all these works (Arora et al., 2012, 2013) were published after the programming portion of this thesis commenced.

5.3 Improvements

For simplicity, elegance, and maintainability, I had to strike a balance between cutting-edge and bleeding-edge. If I had used programming tools that were even more state-of-the-art when I first started this work years ago, I probably would have used the Elastic Stack. Now that it is more widely used, for future work I would advise using it for both monitoring and debugging as well as for visualizing data. Since Elasticsearch uses Lucene, Elasticsearch is especially promising for systems that depend on TF-IDF. However, Elastic Stack can be helpful to any system that requires visual inspection of stopwords.

At the start of this work, nodeJS was a technology which promised to be fast, especially for generating responses to web site requests. Since I initially planned to have the model used by SCI and `http://bean5.github.io/lds-talks-by-topic/`, nodeJS was a language of choice. Although it proved to be difficult and cumbersome and was constantly changing at first—it was bleeding-edge technology at the time—it soon stabilized as it became more widely used (and widely updated). My final framework contains nodeJS code, although personally I believe an open-source TF-IDF project such as Elasticsearch would be an appropriate choice in future work.

In hindsight, it would have made more sense to map inputs at /opt/, while mapping outputs to /var/log/. It would not change results nor run-time, but it would improve readability and maintainability of this project's source code.

The dataset used in this work is already 2 years out-of-date. This is because I did not want to re-manufacture and re-evaluate the module which removes HTML from the corpus. Now that this work is complete, this is certainly a next step before augmenting LDS SCI or similar systems with the LDS-based content.

This work leveraged previous work to select stopwords. This means that stopwords are not dynamically-selected according to the corpus. Therefore, once new documents are added to the corpus, the set should be revisited for possible revision. The best way to do this would be to either use something simple and fast such as AntConc or to use Elasticsearch to query for most frequent words. Using more stopwords will yield run-time improvements to both recommender systems developed here because the search space will be smaller, particularly for the current implementation of TF-IDF.

In reviewing the module that removes HTML, I found the code to be less as elegant than I was hoping it would be. As it turns out, nodeJS, a JavaScript-tied programming language inherits the same disadvantages of built-in regular expression operations—and JavaScript's regular expression engine is not as full-fledged as those of other languages. This makes resultant code more bulky, repetitive, and less elegant. This complaint applies mainly to HTML-based corpora that are inconsistently formatted. Other corpora may not have such problems.

5.4 Contributions

By building a framework for comparing recommender systems, we were able to prove that an LDA RS outperformed a TF-IDF RS for the GC. The framework is flexible: updating or replacing variables, parameters, inputs, outputs, and models is simple. Rather than requiring each module to use the same programming language and shared libraries, we built a system that uses the toolboxes and languages that make the most sense for the operation: Java-based docker images were used for running modules that required the Mallet toolkit while nodeJS was used in other modules. The *RelRec* framework is generalizable to other corpora and runs in under 4 hours on the *GC*.

Bibliography

Anima Anandkumar, Yi-kai Liu, Daniel J Hsu, Dean P Foster, and Sham M Kakade. A Spectral Algorithm for Latent Dirichlet Allocation. In *Advances in Neural Information Processing Systems*, pages 917–925, 2012. 36

Laurence Anthony. Developing AntConc for a new generation of corpus linguists. In *Proceedings of the Corpus Linguistics Conference (CLC 2013)*, pages 14–16, 2013. 5

Sanjeev Arora, Rong Ge, and Ankur Moitra. Learning topic models–going beyond SVD. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 1–10. IEEE, 2012. 36, 37

Sanjeev Arora, Rong Ge, Yonatan Halpern, David M Mimno, Ankur Moitra, David Sontag, Yichen Wu, and Michael Zhu. A Practical Algorithm for Topic Modeling with Provable Guarantees. In *International Conference on Machine Learning (2)*, pages 280–288, 2013. 9, 36, 37

Michael Bean and Eric Ringger. Theological topics through time: An application of Gibbs-sampled LDA and post-hoc metrics to compare religious venues. [Online; accessed on 2014-09-01], December 2013. URL `http://bean5.github.io/research/CLDSGCT-LDA-ToT.pdf`. 18

Joeran Beel and Stefan Langer. A Comparison of Offline Evaluations, Online Evaluations, and User Studies in the Context of Research-Paper Recommender Systems. In *Research and Advanced Technology for Digital Libraries: 19th International Conference on Theory and Practice of Digital Libraries, TPDL 2015, Poznań, Poland, September 14-18, 2015, Proceedings*, pages 153–168. Springer International Publishing, 2015. 11

Joeran Beel, Marcel Genzmehr, Stefan Langer, Andreas Nürnberger, and Bela Gipp. A Comparative Analysis of Offline and Online Evaluations and Discussion of Research Paper Recommender System Evaluation. In *Proceedings of the International Workshop on Reproducibility and Replication in Recommender Systems Evaluation (RepSys '13)*, pages 7–14, New York, NY, USA, 2013a. ACM. 11

Joeran Beel, Stefan Langer, Marcel Genzmehr, Bela Gipp, Corinna Breitinger, and Andreas Nürnberger. Research Paper Recommender System Evaluation: A Quantitative Literature Survey. In *Proceedings of the International Workshop on Reproducibility and Replication in Recommender Systems Evaluation (RepSys '13)*, pages 15–22, New York, NY, USA, 2013b. ACM. 11

Robert M. Bell and Yehuda Koren. Lessons from the Netflix Prize Challenge. *SIGKDD Explorations Newsletter*, 9(2):75–79, December 2007. 6

Andrzej Bialecki. Luke - Lucene Index Toolbox. [Online; accessed on 2013-11-14], November 2013. URL `http://code.google.com/p/luke/`. 5

David M Blei and Michael I Jordan. Variational inference for Dirichlet process mixtures. *Bayesian analysis*, 1(1):121–143, 2006. 9

David M Blei and John D Lafferty. A correlated topic model of science. *The Annals of Applied Statistics*, pages 17–35, 2007. 8

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, March 2003a. 7, 20

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of earch*, 3(Jan):993–1022, 2003b. 8, 10

Jordan L Boyd-Graber, David M Blei, and Xiaojin Zhu. A Topic Model for Word Sense Disambiguation. In *EMNLP-CoNLL*, pages 1024–1033, 2007. 7

Mark Davies. Corpus of Global Web-Based English: 1.9 billion words from speakers in 20 countries. [Online; accessed on 2013-10-24], October 2013. URL `http://corpus.byu.edu/glowbe/`. iii

Elasticsearch. Elasticsearch — elastic, August 2016. URL `https://www.elastic.co/products/elasticsearch`. [Online; accessed on 2016-08-14]. 4

Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. Beyond Accuracy: Evaluating Recommender Systems by Coverage and Serendipity. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 257–260, New York, NY, USA, 2010. ACM. 11, 12, 13, 20, 36

Matthew Hoffman, Francis R. Bach, and David M. Blei. Online Learning for Latent Dirichlet Allocation. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 856–864. Curran Associates, Inc., 2010. 9

John P Huelsenbeck, Sonia Jain, Simon WD Frost, and Sergei L Kosakovsky Pond. A Dirichlet process model for detecting positive selection in protein-coding DNA sequences. *Proceedings of the National Academy of Sciences*, 103(16):6263–6268, 2006. 8

Harold Jeffreys. *Scientific Inference*. Cambridge University Press, 1973. 8

Kriste Krstovski, David A. Smith, Hanna M. Wallach, and Andrew McGregor. Efficient Nearest-Neighbor Search in the Probability Simplex. In *Proceedings of the 2013 Conference on the Theory of Information Retrieval*, ICTIR '13, pages 22:101–22:108, New York, NY, USA, 2013. ACM. 10, 11

Wei Li and Andrew McCallum. Pachinko allocation: DAG-structured mixture models of topic correlations. In *Proceedings of the 23rd international conference on Machine learning*, pages 577–584. ACM, 2006. 8

Inderjeet Mani. Summarization Evaluation: An Overview. 2001. URL `http://research.nii.ac.jp/ntcir/workshop/OnlineProceedings2/sum-mani.pdf`. [Online; accessed on 2016-11-]. 3

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. Scoring, term weighting, and the vector space model. In *Introduction to Information Retrieval*, pages 100–123. Cambridge University Press, 2008. Cambridge Books Online. 5, 6

Andrew Kachites McCallum. MALLET: A Machine Learning for Language Toolkit. Technical report, University of Massachusetts at Amherst, 2002. URL `http://mallet.cs.umass.edu`. 9

Michael McCandless, Erik Hatcher, and Otis Gospodnetic. *Lucene in Action, Second Edition: Covers Apache Lucene 3.0*. Manning Publications Co., Greenwich, CT, USA, 2010. 4, 5

Prem Melville and Vikas Sindhwani. *Recommender Systems*, pages 829–838. Springer US, Boston, MA, 2010. 1

NaturalNode. Natural, August 2016. URL `https://github.com/NaturalNode/natural`. [Online; accessed on 2016-08-09]. 4

Ian Porteous, David Newman, Alexander Ihler, Arthur Asuncion, Padhraic Smyth, and Max Welling. Fast Collapsed Gibbs Sampling for Latent Dirichlet Allocation. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 569–577, New York, NY, USA, 2008. ACM. 9

Jonathan K. Pritchard, Matthew Stephens, and Peter Donnelly. Inference of Population Structure Using Multilocus Genotype Data. *Genetics*, 155(2):945–959, 2000. 7

Stephen Robertson. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of Documentation*, 60(5):503–520, 2004. 6

Alfredo Serafini. *Apache Solr Beginner's Guide*. Packt Publishing Ltd, December 2013. 4

S Shivashankar, S Srivathsan, Balaraman Ravindran, and Ashish V Tendulkar. Multi-view methods for protein structure comparison using latent dirichlet allocation. *Bioinformatics*, 27(13):61–68, 2011. 8

David Smiley and David Eric Pugh. *Apache Solr Enterprise Search Server - Third Edition*. Packt Publishing Ltd, 3 edition, May 2015. 4

Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972. 5

Yee W Teh, David Newman, and Max Welling. A Collapsed Variational Bayesian Inference Algorithm for Latent Dirichlet Allocation. In *Advances in Neural Information Processing Systems*, pages 1353–1360, 2006. 9

Peter D Turney and Patrick Pantel. From Frequency to Meaning: Vector Space Models of Semantics. *Journal of Artificial Intelligence Research*, 37(1):141–188, 2010. 9

Xuerui Wang and Andrew McCallum. Topics over Time: A non-Markov Continuous-time Model of Topical Trends. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 424–433, New York, NY, USA, 2006. ACM. 13

Xing Wei and W. Bruce Croft. LDA-based Document Models for Ad-hoc Retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 178–185, New York, NY, USA, 2006. ACM. 8

Jun Won Lee and Yiu-Kai Ng. Using Fuzzy-Word Correlation Factors to Compute Document Similarity Based on Phrase Matching. In *Fuzzy Systems and Knowledge Discovery, 2007. FSKD 2007. Fourth International Conference on Fuzzy Systems and Knowledge Discovery*, volume 2, pages 186–192. IEEE, 2007. 6, 35

Ho Chung Wu, Robert Wing Pong Luk, Kam Fai Wong, and Kui Lam Kwok. Interpreting TF-IDF Term Weights As Making Relevance Decisions. *ACM Transactions on Information Systems*, 26(3):13:1–13:37, June 2008. 5, 6

Appendix A

All Catalog Coverage Values Measured

	Algorithm	
# of Recommendations	TF-IDF	LDA
1	0.03378	0.03558
2	0.05097	0.06216
3	0.06596	0.08655
4	0.08095	0.11033
5	0.09194	0.13032
6	0.10194	0.14831
7	0.11353	0.16730
8	0.12233	0.18329
9	0.13412	0.20108
10	0.14411	0.21707
11	0.15151	0.23306
12	0.15831	0.24725
13	0.16570	0.26004
14	0.17170	0.27124
15	0.17809	0.28723
16	0.18149	0.30062
17	0.18769	0.31361
18	0.19268	0.32660
19	0.19808	0.33780
20	0.20388	0.34999
21	0.20927	0.36098
22	0.21327	0.36858
23	0.21927	0.37577
24	0.22367	0.38557
25	0.22846	0.39396

Table A.1: Catalog coverage values for all models (k=1 through k=25). Precision is set to 5 significant digits.

	Algorithm	
# of Recommendations	TF-IDF	LDA
26	0.23186	0.40536
27	0.23746	0.41235
28	0.24225	0.41875
29	0.24785	0.42674
30	0.25305	0.43474
31	0.25825	0.44333
32	0.26164	0.44913
33	0.26564	0.45633
34	0.26944	0.46452
35	0.27484	0.47332
36	0.28003	0.48251
37	0.28303	0.48971
38	0.28783	0.49530
39	0.29242	0.50270
40	0.29542	0.50849
41	0.29782	0.51289
42	0.30102	0.51809
43	0.30402	0.52429
44	0.30642	0.53108
45	0.30841	0.53748
46	0.31061	0.54247
47	0.31341	0.54727
48	0.31621	0.55107
49	0.31881	0.55727
50	0.32081	0.56266

Table A.2: Catalog coverage values for all models (k=26 through k=50). Precision is set to 5 significant digits.

	Algorithm	
# of Recommendations	TF-IDF	LDA
51	0.32461	0.56826
52	0.32560	0.57286
53	0.33000	0.57745
54	0.33280	0.58185
55	0.33600	0.58645
56	0.33860	0.59244
57	0.34100	0.59744
58	0.34419	0.60164
59	0.34659	0.60544
60	0.34899	0.61003
61	0.35219	0.61643
62	0.35499	0.61923
63	0.35739	0.62323
64	0.35978	0.62782
65	0.36278	0.63162
66	0.36558	0.63522
67	0.36838	0.63882
68	0.37298	0.64381
69	0.37478	0.64761
70	0.37637	0.65281
71	0.37877	0.65641
72	0.38077	0.65960
73	0.38277	0.66480
74	0.38517	0.66780
75	0.38797	0.67080

Table A.3: Catalog coverage values for all models (k=51 through k=75). Precision is set to 5 significant digits.

	Algorithm	
# of Recommendations	TF-IDF	LDA
76	0.39157	0.67460
77	0.39456	0.67699
78	0.39616	0.68019
79	0.39876	0.68299
80	0.40136	0.68559
81	0.40296	0.68699
82	0.40496	0.68899
83	0.40616	0.69198
84	0.40855	0.69398
85	0.41075	0.69738
86	0.41395	0.70078
87	0.41555	0.70438
88	0.41615	0.70658
89	0.41795	0.70937
90	0.42035	0.71117
91	0.42295	0.71357
92	0.42574	0.71557
93	0.42754	0.71797
94	0.42914	0.72077
95	0.43294	0.72337
96	0.43494	0.72497
97	0.43694	0.72636
98	0.43994	0.72956
99	0.44273	0.73176
100	0.44493	0.73576

Table A.4: Catalog coverage values for all models (k=76 through k=100). Precision is set to 5 significant digits.

Appendix B

Tools Used for This Thesis

Tool/Library	*Use*	*Link(s)*
Git	code versioning; code sharing	`https://git-scm.com/about`
Latex, Google Docs, Microsoft Office	Composing versions of this document	`https://www.latex-project.org/`, `https://docs.google.com`, `https://products.office.com/en-us/word`
AntConc	Indexing of files to discovery high-frequency words (stop-words).	`http://www.laurenceanthony.net/software/antconc/`
"docker" (docker daemon, docker-compose, docker containers)	Containerization and modularization of each module such that when running, code for each module only contains input, the necessary code, output, and operating system tools.	`https://www.docker.com/`
Docker For Mac	Running docker daemon in hypervisor VMs	`https://www.virtualbox.org/`, `https://docs.docker.com/engine/installation/mac/`
SCI database (mysql)	Input into first module	(link not available)
MySQL server, MySQL client	hosting + querying of SCI database	`http://dev.mysql.com/`
GNU Make	orchestration of modules (i.e. the command make thesis runs each module that hasnt been run, making sure to do so in the correct order	`https://www.gnu.org/software/make/`

Table B.1: Tools and libraries used.

Tool/Library	*Use*	*Link(s)*
nodeJS, bash, sh, zsh	orchestration of modules (i.e. the command make thesis runs each module that hasnt been run, making sure to do so in the correct order	`https://nodejs.org/en/`, `https://www.gnu.org/software/bash/`
npm	hosting of open-source nodeJS code	`https://www.npmjs.com/`
Atom, vi	code editors	`https://atom.io/`, `http://www.vim.org/`
Mallet	indexing of files in preparation for LDA; Gibbs Sampling to infer parameters of LDA topic model	`http://mallet.cs.umass.edu/`
Luke (Lucene Index Toolbox)	ranking words by frequency to identify stopwords	`http://www.getopt.org/luke/`
Mac OS-X	Host system running VM, hosting code	`http://www.apple.com/osx/`
Bitbucket	code hosting	`https://bitbucket.org/`
GitHub	code hosting; location of open-source tools used in the project including nodeJS and Mallet	`https://github.com/`
less, more, cat, wc, grep, egrep, z, oh-my-zsh	general modification or searching of files in *nix	`https://en.wikipedia.org/wiki/Less_(Unix)`, `http://linux.die.net/man/1/more`, `http://linux.die.net/man/1/grep`, `http://linux.die.net/man/1/egrep`, `http://linux.die.net/man/1/wc`, `https://github.com/rupa/z`, `https://github.com/robbyrussell/oh-my-zsh`
regular expressions	easy removal of boiler plate HTML from documents	`https://en.wikipedia.org/wiki/Regular_expression`

Table B.2: Continuation of Table B.1.