

**SEMANTIC INFORMATION RETRIEVAL BASED ON
TOPIC MODELING AND COMMUNITY INTERESTS
MINING**

R.P. Minuri Chathurika Rajapaksha

(158734M)

Degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics

University of Moratuwa

Sri Lanka

March 2019

**SEMANTIC INFORMATION RETRIEVAL BASED ON
TOPIC MODELING AND COMMUNITY INTERESTS
MINING**

R.P. Minuri Chathurika Rajapaksha
(158734M)

Dissertation submitted in partial fulfillment of the requirements for the degree Master of
Science in Artificial Intelligence

Department of Computational Mathematics
University of Moratuwa
Sri Lanka

March 2019

DECLARATION

I declare that this dissertation does not incorporate, without acknowledgment, any material previously submitted for a Degree or a Diploma in any University and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis/dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Name of Student

Signature of Student

Minuri Rajapaksha

Date:

The above candidate has carried out research for the Masters Dissertation under my supervision.

Supervised by

Signature of Supervisor

Dr. Thushari Silva

Date:

Abstract

Search engines or localized software systems developed for information searching, play an important role in knowledge discovery. Proliferation of data in the web and social media has posed significant challenges in finding relevant information efficiently even using those search engines or other software systems. Moreover, those systems or engines tend to collect massive number of data, which could be useful for humans in various ways but overlook the meaning of the search phrases, hence generate irrelevant search results. A unit level searching i.e. searching information within a website or page is also not effective as they follow exact keyword matching techniques and ignore the semantic level matching of search phrases. In order to address those deficiencies, this research proposes a hybrid approach which use the semantics of data, community preferences as well as collaborative filtering techniques for semantic information retrieval. More specifically, Topic modeling based on Latent Dirichlet Allocation together with topic-driven based community detection methods are applied for identifying personalized search results and hence improve the relatedness of the research results. Based on the proposed hybrid approach a framework for semantic search that can easily be integrated to a software application has been implemented. The evaluation results confirm the effectiveness of search results which outperform benchmark approaches that follow traditional keyword search algorithms.

TABLE OF CONTENTS

DECLARATION	iii
ABSTRACT	iv
TABLE OF CONTENTS	v
LIST OF FIGURES.....	viii
LIST OF TABLES	x
CHAPTER 1.....	1
1. INTRODUCTION	1
1.1. Prolegomena	1
1.2. Objectives	2
1.3. Background and Motivation	3
1.4. Problem in Brief.....	3
1.5. Proposed Solution	3
1.6. Resource Requirements	5
1.7. Structure of the Thesis	5
1.8. Summary	5
CHAPTER 2.....	6
2. SEMANTIC INFORMATION RETRIEVAL ISSUES AND CHALLENGES .6	
2.1. Introduction.....	6
2.2. Generation of Semantic Information Retrieval.....	6
2.3. State of the Art of Personalized Semantic Information Retrieval.....	8
2.4. Future Trends	16
2.5. Problem Definition.....	16
2.6. Summary	17
CHAPTER 3.....	18
3. TECHNOLOGIES ADAPTED	18
3.1. Introduction.....	18
3.2. Discover Topics Based on Optimized LDA Model.....	18

3.3.	Community Detection Based on Topic-Driven Approach.....	21
3.4.	Collaborative Filtering Module.....	22
3.5.	Search Optimization Module	26
3.6.	Summary	27
CHAPTER 4.....		28
4.	APPROACH.....	28
4.1.	Introduction.....	28
4.2.	Hypothesis.....	28
4.3.	Process	30
4.4.	Input and Output	31
4.5.	Users	31
4.6.	Features	32
4.7.	Summary.....	32
CHAPTER 5.....		33
5.	DESIGN	33
5.1.	Introduction.....	33
5.2.	Discover Topics Module based on Optimized LDA Model	34
5.3.	Community Extraction Module based on Topic-Driven Model	35
5.4.	Item-based Collaborative Filtering Module.....	38
5.5.	Search Optimization Module	39
5.6.	Summary	39
CHAPTER 6.....		40
6.	IMPLEMENTATION	40
6.1.	Introduction.....	40
6.2.	Data Preprocessing.....	42
6.3.	Discover Topics Based on Topic Modeling.....	43
6.3.	Community Detection Based on Topic-Driven Approach.....	44
6.4.	Item-Based Collaborative Filtering Module	46
6.5.	Search Optimization Module	47

6.6. Summary.....	52
CHAPTER 7.....	53
7. EVALUATION	53
7.1. Introduction.....	53
7.2. Experimental Setup.....	53
7.3. Participants.....	53
7.4. Test Cases	53
7.5. Testing Strategies.....	54
7.6. Training Data	54
7.7. LDA Model Performance	54
7.8. Community Detection.....	57
7.9. Search Optimization.....	60
CHAPTER 8.....	62
8. CONCLUSION AND FURTHER WORK	62
8.1. Introduction.....	62
8.2. Conclusion	63
8.3. Further Work.....	65
REFERENCES.....	66
Appendix A – Python Code for Data Preprocessing.....	69
Appendix B – Calculate Distance between users Using Jensen Shannon Divergence	73
Appendix C – Construct the Communities using Girwan Newman Algorithm.....	76
Appendix D – Item Based Collaborative Filtering which returns Recommended Articles for Particular User	80
Appendix E – Integrate with Lucene.....	87

LIST OF FIGURES

Figure 3.1 - LDA model.....	20
Figure 3.2 - Lucene in a search system.....	26
Figure 4.1 - Design diagram (Top level architecture).....	31
Figure 5.1 - High level architecture of semantic information retrieval.....	33
Figure 5.2 - LDA model.....	34
Figure 6.1 - Semantic Information Retrieval System.....	41
Figure 6.2 - Word frequency dictionary.....	42
Figure 6.3 - LDA Model	43
Figure 6.4 - Sample Graph.text file content	45
Figure 6.5 - Constructed communities	46
Figure 6.6 - User Sign-up page	48
Figure 6.7 - User sign-in page.....	48
Figure 6.8 - List of user uploaded files and user can edit uploaded files.....	49
Figure 6.9 - File up loader.....	49
Figure 6.10 - Create new file.....	50
Figure 6.11 - Search results for text “carrom”	50
Figure 6.12 - Search results for text “mom”	50
Figure 6.13 - Search results for text “county”.....	51
Figure 6.14 - Configuration set-up page	51
Figure 6.15 - Uploaded file content	52
Figure 7.1 - Choosing optimal model with coherence scores	56

Figure 7.2 - Topic-word distribution.....	56
Figure 7.3 - Word count and importance of topic keywords	57
Figure 7.4 - Constructed graph.....	58
Figure 7.5 - Topics treated in community1	58

LIST OF TABLES

Table 2.1 - Topic Modeling Current Practices and Issues9

Table 2.2 - Community Detection Current Practices And Issues 10

Table 7.1 - Evaluate the model basaed on different topic numbers55

Table 7.2 - Distance between users57

Table 7.3 - Closeness between users for selected topic58

Table 7.4 - Evaluate the community detection based on different number of topics for trained model.....59

Table 7.5 - Filter Articles User Uploaded Wise..... 60

Table 7.6 - Filter articles based on item based collaborative filtering 61

Table 7.7 - Prioritize search results 61

CHAPTER 1

1. INTRODUCTION

1.1. Prolegomena

There is a great growing demand for sophisticating information retrieval based on users' need with the help of search engines [1]. To meet this demand current system heavily used data representation technique in computer systems including database models, especially relational databases which enables efficient information storing and querying [2]. Since most of the current web applications are supporting relational schema-based information retrieval, they could not provide any semantic sense of feedback to users. Now ontologies emerged as an alternative to databases in applications that require a more 'enriched' meaning [3].

Today's web applications contain vast amount of data and users of those applications are looking for easy access to the data they want in a short period of time rather wasting time searching bulk of data. Those applications contain large number of documents also and handling and retrieval should be efficient for obtaining a business value. Almost all search engines use text-based search techniques where it matches the query string with the text in the files or database. The search result is generated just based on the number of occurrences and this does not take real meaning of the query string. The same applies to an application where the user tries to find help details inside an application. Hence searching based on content only has become a challenge to most applications [4].

One of the primary applications of natural language processing is to automatically extract what topics people are discussing from large volumes of text. Some examples of large text are feeds from social media, customer reviews of hotels, movies, etc. user feedbacks, news stories, e-mails of customer complaints etc. Knowing what people are talking about and understanding their problems and opinions is highly valuable to businesses, administrators, political campaigns. And it's really hard to manually read

through such large volumes and compile the topics. Thus, is required an automated algorithm that can read through the text documents and automatically output the topics discussed. In such a case, semantic search strategies could enhance semantic search results.

Even though sites like “Google” provides better search results with personalization up to some extent, in general individual sites lack semantic search engines to search within its site. Intelligent Semantic Frameworks for individual sites [5] has focused only to Latent Semantic Indexing and Personalization based on users’ history. Therefore, current semantic search engines lack a model which focus on semantics of data, community preferences based on dynamically changing semantics of data and personalization. Therefore, the proposed solution is to provide internal semantic search engine to search within its site based on semantics of data and community interests. The proposed model is more focusing on better semantics of data extraction based on LDA model and community detection based on LDA model to extract communities from the semantics of data itself dynamically. Further the model has optimized based on personalization.

1.2. Objectives

- a) In-depth study of technology used for semantic meta data extraction.
- b) Critical review of analyzing ‘item based collaborative filtering’.
- c) Critical review of analyzing ‘topic-based community detection’.
- d) Design and develop semantic information retrieval platform
- e) Evaluate solution
- f) Document the entire project

1.3. Background and Motivation

Today world contains lots of accessible information pieces, which could be unreadable by machines, but could be understandable by humans. To be able to use this information people are heavily engaged with search engines. Between key search and semantic search, semantic search plays a major role in semantic information retrieval. Lack of semantic search framework which can directly integrate with industry software applications is the key motivational point towards this research. Proving semantically relevant search results and personalized semantic search based on dynamically detected communities would be really beneficial for each web application today. Even though search engine like “Google” provide better search results based on personalized results, most individual sites lack internal semantic search engines. Most of the individual software applications lacks internal semantic search engines within their sites.

1.4. Problem in Brief

Current software industry application search engines support mostly just relational data-based and text-based information retrieval. It lacks semantic information retrieval platform, which provides user most appropriate search results based on relevancy and personalization. Those Search platforms lack providing search results based on most relevant topics for a given context and based on dynamically detection of individuals who are sharing common topic interests in a community.

1.5. Proposed Solution

Proposed solution is to provide a semantic information retrieval platform for internal searching of a software application. Information retrieval would be based on semantics of data, based on topic-based community detection, based on applying personalization and collaborative filtering on top of detected communities for providing more meaningful search results to user. To provide semantically rich search results, application will look into latent Dirichlet allocation and topic-driven community detection methods integrating with collaborative filtering.

The system contains five modules called

1. UI module which provide user interface for uploading documents and entering search query. Here user can create documents and folders also. User can view, edit, delete uploaded files. User can configure the settings related to their company.
2. Discover topics Module which discover hidden topics in user uploaded documents. First the documents are preprocessed and then train LDA model and by using LDA model do topic extraction. These extracted topics are used to define communities and indexing search engine.
3. Community extraction module which is detecting communities based on topic-driven approach and community detection algorithms. Here by using LDA topic-model which returned user-topic distributions and by using Jensen-Shannon Divergence algorithm, the topic distances between two users are calculated. Then using the calculated distances, community graph has been constructed. Then to construct use communities used an adapted approach of Girvan-Newman which is based on divisive classification.
4. Item based collaborative filtering Module which uses the detected communities and user history to find more relevant articles. Item based collaborative filtering consider about the user search history in a particular community (based on communities identified by above community detection module) and according to the frequency of any article viewed or downloaded, article would be given a preference value. These inferred preference values are stored with the search results and they are used to derive a final composite score, on which ultimate search results are based on.
5. Search optimization Module which is responsible for integrating above modules to search engine, build indexing, rank files, execute the query and find best search results. Lucene is used to internalize the topics and topic memberships while building the index and executing the queries. Here used Payloads to cleverly encode the topics in each document at index time. When user has entered the query, determine which topics are in the query, based on the terms in the query. Then create

a Payload query based on these topics. Lucene will then find all documents that contain these topics. We ignore the actual relevancy returned by Lucene, and instead use the contents of the Payload to compute the relevancy ourselves, and re-rank the results.

1.6. Resource Requirements

1. Software application, which stores document repository, which calls APIs, which calls libraries, which implements algorithms and which integrates all modules into one.
2. Programming Languages, client side scripting languages (ASP.Net Core, HTML, CSS, Java script), databases(SQL Server).
3. Server for storing files and to run software application.

1.7. Structure of the Thesis

This thesis has been structured with 8 chapters. Chapter 1 gave an overall introduction the project. Chapter 2 provides a critical review of the developments and issues in the area of semantic information retrieval by defining the research problem and identification of technologies. Chapter 3 is on technology adapted to semantic information retrieval platform. Chapter 4 presents our approach to semantic information retrieval for software industry applications. Chapter 5 illustrates the design of research and Chapter 6 presents the how the application is implemented. Chapter 7 describes how the application is evaluated based on several criteria. Finally, Chapter 8 concludes the thesis with a note on the possible further work.

1.8. Summary

This chapter provided an introduction to the entire project. For this purpose, we have presented our research problem, objectives, technology adopted, proposed solution and resource requirements. Next chapter provides a detailed critical review of semantic information retrieval system.

CHAPTER 2

2. SEMANTIC INFORMATION RETRIEVAL ISSUES AND CHALLENGES

2.1. Introduction

Chapter 1 gave an introduction to overall project. This chapter presents a critical review of literature on semantic information retrieval.

The study of relationships between words and how we construct meaning, how we experience the world, how we understand others and ourselves can be considered as semantic language. Semantics can also refer to the branch of study within linguistics that deals with language and how we understand meaning. Hence searching through documents considering more than one syntactic level of keyword matching can be called as semantic search. This plays a major role in semantic searching unlike key word matching in traditional searching. In traditional searching if it fails to catch exact key word, then it would fail to provide any result. In searching based on semantic data, to provide actual meaning of data, the semantics of data should be considered where ever it is possible. Semantic languages are used to keep track of metadata of a data source.

2.2. Generation of Semantic Information Retrieval

Searching technology of the first generation that regards directory index as characteristics keywords and the second generation considers the importance of pages of searching technology of technical analysis and hyperlink features. Network information is developing towards the third generation searching technology with semantic and personality.

The first generation of searching technology product is yahoo, data model of network literature is established and its model is similar to the approach of the library literature classification, the function is simple and not standardized. During the course of manual

classification, there exist the problems of high cost and low efficiency, which does not satisfy the rapid growth of online information resources management.

The second generation of searching technology is google and baidu in domestic products, both data models of network resource use a number of keywords to express, and the inverted index is established. This model has the advantages that it can provide large scale searching for the computers, but keywords are only symbol of the page in it, and the semantic which is referred has not been used. As the searching technology cannot understand the semantics which exists at the website of the information, leading to much duplication and too little useful information, searching information is inconvenient.

The third generation searching technology uses semantic data model that combines xml, rdf, ontology, laying the foundation for the content that computer can learn, now there are not representation products. The current searching technology is mainly based on matching keywords or full text of classification based on themes. The results often return a large number of unrelated links, allowing user to spend too much time on the exclusion of irrelevant information. At the same time users and network documentation on the presentation of the same concept are different, then does not receive useful information.

In order to solve the current problems of information retrieval, the searching technology is more effective tool that is firstly applied to the network information searching. But there are some defects such as providing search results based on dynamically detected community interests, poor real-time information, searching technology in the database cannot be changed with the network of dynamic documents, which is unable to provide timely updated information for the users, information seekers are poor and cannot retrieve the corresponding areas of information according to the requirements of users, leading to emergence of large number of irrelevant information, and frequently using searching technology every day for millions of users with services, even if such searching technology that uses high performance server hardware systems is also difficult to provide such a range of rapid response.

2.3. State of the Art of Personalized Semantic Information Retrieval

Search engines are based on several approaches. Two major principles we could identify in search engines. It is key search and semantic search.

- Key search

It is based on just keywords or phrases. People use technologies like tokenizers, filters etc. it uses process called indexation which is for transforming input document into one term. It is the base for searching. The issue with that is they don't consider semantics like relationships, knowledge etc.

- Semantic search

This technology is based on the semantics of the data. It is more intelligent than key searching. It could use synonyms in general way to find hidden some information.

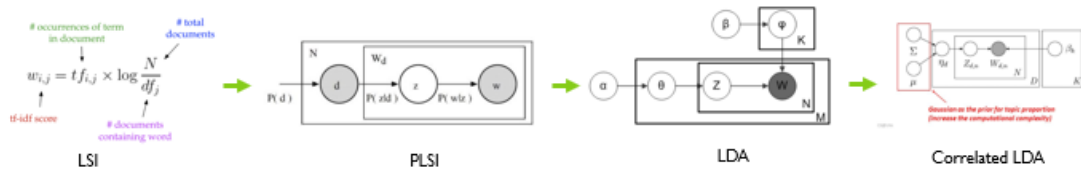
One of the targets of W3C consortium is also semantic web. It provides styles for web which could be readable understandable by both humans and machines. Therefore, machine knows how to find, infer or derive the information.

A. Topic Modeling

The predecessor of topic modeling can be traced back to LSA (Latent Semantic Analysis) [6]. LSA is based on spatial dictionaries, and implicit semantic documents are implemented in low- dimensional representation of space, but it cannot solve the problem of the coexistence of many possible meanings for a word or phrase. Hofmann proposes a PLSA (Probabilistic Latent Semantic Analysis) [7,8] for the defect of LSA, mainly using the probability distribution corresponding to one dictionary in each dimension. However, PLSA does not provide a probabilistic model at the document level, which leads to overfitting problems easily due to the linear increase in the number of parameters to be estimated in the model with the size of the corpus. LDA (Latent Dirichlet Allocation) [9] is a generation model that uses the Dirichlet a priori distribution of topics to overcome the shortcomings of PLSA. The model can find the semantic structure of the text set, mining the theme of the text (Table 2.1).

Table 2.1 - Topic Modeling Current Practices and Issues

Topic Models	High Performance	Accuracy	Heuristic	Likelihood	Bayesian	Correlation
LSI			✓			
Probabilistic LSI				✓		
LDA	✓	✓			✓	
Correlated LDA	✓	✓			✓	✓



B. Community Detection

Community detection algorithms has been widely used for detecting hidden communities. They can be generally divided into two classes: topic-based approaches and structure-based approaches. Widely studied structure-based approach [11,12] does not have a clear perspective on how to make sense of identified communities. Only a few research efforts fall into the topic-based approach, which groups individuals sharing common topic interests into a community.

Sachan et al., [18] proposed a generative model to discover communities based on topics, social graph topology, and nature of user interactions. Pathak et al., [19] proposed a Bayesian generative model for community extraction which considered both the network topology and user topic to generate communities. Structure-based approaches for detecting communities mine shared common interests on Twitter based on their relationship hierarchies starting from celebrities which represents an interest category [20]. A LDA-based model has been used to detect user topics based on their tweets [21], then created the topic graph also called semantic graph where the weight was the topic similarity of two users, then applied existing community detection algorithm to find the community in the topic graph. Jaffali et al., [22] presented an approach based on grouping users who share the same interests by analyzing their textual posts. They applied Principal Component Analysis to find the principle components, called interest

center and used k-mean clustering algorithms to cluster the users based on their distance to principle components. Current semantic search engines lack a dynamic approach which uses the user uploaded files for detecting communities (Table 2.2).

Table 2.2 - Community Detection Current Practices And Issues

Models	Network topology	Community detection algorithm	K-means	Principal Component Analysis	Topic analysis	Sentiment Analysis	LDA	JSD
Sachan et al	✓		✓					
Structure based	✓				✓			
Pathak et al	✓							
Jaffali et al			✓	✓		✓		
Topic-Driven		✓				✓	✓	✓

C. Collaborative Filtering

Personalization is essential for personalized search results suggestion. Providing search results based on user preferences as well as their profiles is one way of doing personalization and it is tightly coupled with the relevant search functionality. Collaborative filtering which is massively used in recommendation systems is an approach deriving profile based on who historically had similar tastes [13]. Current semantic search engines' personalization is based on user history and already defined communities, but they lack dynamically detected community-based personalization approach.

Item based collaborative filtering is much popular by now than user based collaborative filtering. Item based collaborative filtering which was presented by Badrul Sarwar [14] has overcome scalability problems and literature emphasizes is better than K-nearest algorithm (Table 2.3).

Table 2.3 – Collaborative filtering current practices and issues

Models	Examples	Speed	Static relationship	Easy to implement	Scalability	Less online computation	Flexibility	Quality of prediction
Item(model) based	Bayesian Network, Clustering, LDA Model	✓	✓		✓	✓		
Memory based	Minkovski Distance, Pearson Correlation, Cosine metrics use to calculate similarity			✓			✓	✓

Information retrieval by searching information on the web is not a fresh idea but has different challenges when it is compared to general information retrieval. Different search engines return different search results due to the variation in indexing and search process. Google, Yahoo, and Bing have been out there which handles the queries after processing the keywords. They only search information given on the web page, recently, some research group's start delivering results from their semantics based search engines, and however most of them are in their initial stages. Till none of the search engines come to close indexing the entire web content, much less the entire Internet. Current web is the biggest global database that lacks the existence of a semantic structure and hence it makes difficult for the machine to understand the information provided by the user. When the information was distributed in web, we have two kinds of research problems in search engine i.e.

- How can a search engine map a query to documents where information is available but does not retrieve in intelligent and meaning full information. The query results produced by search engines are distributed across different documents that may be connected with hyperlink.
- How search engine can recognize efficiently such a distributed results.

Semantic web [27] [28], can solve the first problem in web with semantic annotations to produce intelligent and meaningful information by using query interface

mechanism and ontology's. Other one can be solved by the graph-based query models [29]. The Semantic web would require solving extraordinarily difficult problems in the areas of knowledge representation, natural language understanding. The following figure depicts the semantic web frame work it also referred as the semantic web layercake by W3C.

Present World Wide Web is the longest global database that lacks the existence of a semantic structure and hence it becomes difficult for the machine to understand the information provided by the user in the form of search strings. As for results, the search engines return the ambiguous or partially ambiguous result data set; Semantic web is being to be developed to overcome the following problems for current web.

- The web content lacks a proper structure regarding the representation of information.
- Ambiguity of information resulting from poor interconnection of information.
- Automatic information transfer is lacking.
- Usability to deal with enormous number of users and content ensuring trust at all levels.
- Incapability of machines to understand the provided information due to lack of a universal format.

Hakia [30] is a general purpose semantic search engine that search structured text like Wikipedia. Hakia calls itself a “meaning-based (semantic) search engine” [31]. They're trying to provide search results based on meaning match, rather than by the popularity of search terms. The presented news, Blogs, Credible, and galleries are processed by hakia's proprietary core semantic technology called QDEXing [30]. It can process any kind of digital artifact by its Semantic Rank technology using third party API feeds [32].

D. Types of Semantic Search Engines

Semantic is the process of communicating enough meaning to result in an action. A sequence of symbols can be used to communicate meaning, and this communication can then affect behavior. Semantics has been driving the next generation of the Web as the Semantic Web, where the focus is on the role of semantics for automated approaches to exploiting Web resources. 'Semantic' also indicates that the meaning of data on the web can be discovered not just by people, but also by computers. Then the Semantic Web was created to extend the web and make data easy to reuse everywhere.

Semantic web is being developed to overcome the following main limitations of the current Web [33]:

- The web content lacks a proper structure regarding the representation of information.
- Ambiguity of information resulting from poor interconnection of information.
- Automatic information transfer is lacking.
- Unable to deal with enormous number of users and content ensuring trust at all levels.
- Incapability of machines to understand the provided information due to lack of a universal format.

✓ Semantic search engines

Currently many of semantic search engines are developed and implemented in different working environments, and these mechanisms can be put into use to realize present search engines.

Alcides Calsavara and Glauco Schmidt proposes and defines a novel kind of service for the semantic search engine. A semantic search engine stores semantic information about Web resources and is able to solve complex queries, considering as well the context where the Web resource is targeted, and how a semantic search engine may be employed

in order to permit clients obtain information about commercial products and services, as well as about sellers and service providers which can be hierarchically organized [34]. Semantic search engines may seriously contribute to the development of electronic business applications since it is based on strong theory and widely accepted standards.

Sara Cohen Jonathan Mamou et al presented a semantic search engine for XML (XSEarch) [35]. It has a simple query language, suitable for a naïve user. It returns semantically related document fragments that satisfy the user's query. Query answers are ranked using extended information-retrieval techniques and are generated in an order similar to the ranking. Advanced indexing techniques were developed to facilitate efficient implementation of XSEarch. The performance of the different techniques as well as the recall and the precision were measured experimentally. These experiments indicate that XSEarch is efficient, scalable and ranks quality results highly.

Bhagwat and Polyzotis propose a Semantic-based file system search engine- Eureka, which uses an inference model to build the links between files and a File Rank metric to rank the files according to their semantic importance [36]. Eureka has two main parts: a) crawler which extracts file from file system and generates two kinds of indices: keywords' indices that record the keywords from crawled files, and rank index that records the File Rank metrics of the files; b) when search terms are entered, the query engine will match the search terms with keywords' indices, and determine the matched file sets and their ranking order by an information retrieval- based metrics and File Rank metrics.

Wang et al . project a semantic search methodology to retrieve information from normal tables, which has three main steps: identifying semantic relationships between table cells; converting tables into data in the form of database; retrieving objective data by query languages [37]. The research objective defined by the authors is how to use a given table and a given domain knowledge to convert a table into a database table with semantics. The authors' approach is to denote the layout by layout syntax grammar and match these denotation with given templates which can be used to analyze the semantics

of table cells. Then semantic preserving transformation is used to transform tables to database format.

Kandogan et al . develop a semantic search engine-Avatar, which combines the traditional text search engine with use of ontology annotations [37]. Avatar has two main functions: a) extraction and representation – by means of UIMA framework, which is a workflow consisting of a chain of annotators extracted from documents and stored in the annotation store; b) interpretation – a process of automatically transforming a keyword search to several precise searches. Avatar consists of two main parts: semantic optimizer and user interaction engine. When a query is entered into the former, it will output a list of ranked interpretations for the query; then the top- ranked interpretations are passed to the latter, which will display the interpretations and the retrieved documents from the interpretations.

✓ Ontology search engines

Maedche et al . designed an integrated approach for ontology searching, reuse and update [38]. In its architecture, an ontology registry is designed to store the metadata about ontologies and ontology server stores the ontologies. The ontologies in distributed ontology servers can be created, replicated and evolved. Ontology metadata in ontology registry can be queried and registered when a new ontology is created. Ontology search in ontology registry is executed under two conditions -query-by-example is to restrict search fields and search terms, and query- by-term is to restrict the hyponyms of terms for search.

Georges Gardarin et al. discussed a SEWISE [39] is an ontology-based Web information system to support Web information description and retrieval. According to domain ontology, SEWISE can map text information from various Web sources into one uniform XML structure and make hidden semantic in text accessible to program. The textual information of interest is automatically extracted by Web Wrappers from various Web sources and then text mining techniques such as categorization and summarization are used to process retrieved text information.

2.4. Future Trends

Topic extraction based on topic models, community detection based on topic-driven approach and collaborative filtering have now become emerging areas by now. Providing search results based on discovered topics, based on community user preferences is one way of enhancing semantic search results and it is tightly coupled to search functionality.

Emulating brain learning structure based on the Random Neural Network with Deep Learning clusters [8] has emerge as a semantic information retrieval mechanism for large data sets today. This algorithm measures and evaluates Web result relevance by assigning each Deep Learning cluster with a specific Web Search Engine and it selects the best performing learning cluster to teach other clusters. The learning clusters outperform other Web search engines and we confirm that cluster performance can be improved by learning from best learning clusters [9].

2.5. Problem Definition

Today the web applications contain vast amount of data which contain large number of documents also. Most web application search engines support mostly just relational data-based and text-based information retrieval. It lacks semantic information retrieval platform, which provides user most appropriate search results based on relevancy and personalization. Almost all search engines use text-based search where it matches the query string with the text in the files or database. The search result is generated just based on the number of occurrences and this does not take real meaning of the query string. The same applies to an application where the user tries to find help details. Those Search platforms lack providing search results based on most relevant topics for a given context and based on individuals who are sharing common topic interests in a community.

Present web application semantic search engines lack providing more semantically rich and personalize search results. The search engines use user history and already

established communities (eg: twitter followers topology) [5*] for personalizing data. Detecting communities from the semantic content itself is hardly found in literature.

1. Limitations

- Semantic search engines lack of dynamic community detection approach which uses the current semantics of data
- Incapability of finding user community preferences, when established community structure is not given
- Inability to extract more relevant semantics of data
- Semantic search engines lack a model combined with semantics of data extraction using latest technology (LDA) module and dynamic community extraction module.
- Hence searching based on content has become a challenge to most applications.

2.6. Summary

Chapter 2 described about the background towards this research. The generation of semantic information retrieval and future trends towards semantic information retrieval. Chapter 2 specifically described about the Topic modeling, Community detection and Collaborative filtering current practices and issues. Next chapter will discuss about the technologies adapted. .

CHAPTER 3

3. TECHNOLOGIES ADAPTED

3.1. Introduction

Proposed solution is to provide a semantic information retrieval platform for internal searching of a software web application. Information retrieval would be based on topics extracted from topic-model, based on communities detected from topic-driven model and based on applying personalization and collaborative filtering for providing more meaningful search results to user. To provide semantically rich search results, application will look into Latent Dirichlet Allocation, Community detection algorithms, indexing and ranking methods integrating with collaborative filtering.

3.2. Discover Topics Based on Optimized LDA Model

At the document level, one of the most useful ways to understand text is by analyzing its topics. The process of learning, recognizing, and extracting these topics across a collection of documents is called topic modeling. Topic modeling most popular techniques today are: LSA, pLSA, LDA, and the newer, deep learning-based lda2vec.

All topic models are based on the same basic assumption:

- each document consists of a mixture of topics, and
- each topic consists of a collection of words.

In other words, topic models are built around the idea that the semantics of our document are actually being governed by some hidden, or “latent,” variables that we are not observing. As a result, the goal of topic modeling is to uncover these latent variables-topics-that shape the meaning of our document and corpus. The rest of this blog post will build up an understanding of how different topic models uncover these latent topics.

- LSA

Latent Semantic Analysis, or LSA, is one of the foundational techniques in topic modeling. The core idea is to take a matrix of what we have—documents and terms—and decompose it into a separate document-topic matrix and a topic-term matrix.

The first step is generating our document-term matrix. Given m documents and n words in our vocabulary, we can construct an $m \times n$ matrix A in which each row represents a document and each column represents a word. In the simplest version of LSA, each entry can simply be a raw count of the number of times the j -th word appeared in the i -th document. In practice, however, raw counts do not work particularly well because they do not account for the significance of each word in the document. For example, the word “nuclear” probably informs us more about the topic(s) of a given document than the word “test.”

- PLSA

pLSA, or Probabilistic Latent Semantic Analysis, uses a probabilistic method instead of SVD to tackle the problem. The core idea is to find a probabilistic model with latent topics that can generate the data we observe in our document-term matrix. In particular, we want a model $P(D,W)$ such that for any document d and word w , $P(d,w)$ corresponds to that entry in the document-term matrix.

Recall the basic assumption of topic models: each document consists of a mixture of topics, and each topic consists of a collection of words. pLSA adds a probabilistic spin to these assumptions:

given a document d , topic z is present in that document with probability $P(z|d)$

given a topic z , word w is drawn from z with probability $P(w|z)$.

- LDA

LDA stands for Latent Dirichlet Allocation. LDA is a Bayesian version of pLSA. In particular, it uses dirichlet priors for the document-topic and word-topic distributions, lending itself to better generalization.

Consider the very relevant example of comparing probability distributions of topic mixtures. Let's say the corpus we are looking at has documents from 3 very different subject areas. If we want to model this, the type of distribution we want will be one that very heavily weights one specific topic, and doesn't give much weight to the rest at all. If we have 3 topics, then some specific probability distributions we'd likely see are:

- Mixture X: 90% topic A, 5% topic B, 5% topic C
- Mixture Y: 5% topic A, 90% topic B, 5% topic C
- Mixture Z: 5% topic A, 5% topic B, 90% topic C

If we draw a random probability distribution from this dirichlet distribution, parameterized by large weights on a single topic, we would likely get a distribution that strongly resembles either mixture X, mixture Y, or mixture Z. It would be very unlikely for us to sample a distribution that is 33% topic A, 33% topic B, and 33% topic C.

That's essentially what a dirichlet distribution provides: a way of sampling probability distributions of a specific type. Recall the model for pLSA:

The model the proposed system use, is the well-known statistical topic model, Latent Dirichlet Allocation (LDA) [10,11]. In LDA model (Figure 3.1), the hidden semantic structure includes:

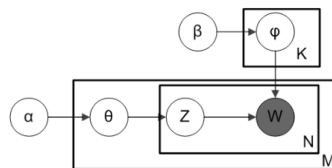


Figure 3.1 - LDA model

3.3. Community Detection Based on Topic-Driven Approach

A number of community-detection algorithms and methods have been proposed and deployed for the identification of communities in literature. There have also been modifications and revisions to many methods and algorithms already proposed. We can divide the previous work in literature in two approaches

- ✓ First Approach (Based On The Relation Between Community Members)

This approach divided into four categories. First category is node centric community detection, Second category is group centric community detection (Density-Based Groups), and third category is network centric community detection, while the fourth category is hierarchy centric community detection.

- ✓ Node centric community detection

Wherever nodes satisfy different properties as complete mutuality which implies cliques; another property is reachability of members as k-clique.

- ✓ Group centric community detection (Density-Based Groups)

It needs the total group to satisfy an explicit condition for instance the group density greater than or equal a given threshold and take away nodes with degree under the typical average degree.

- ✓ Network centric community detection

It takes into account connections within a network globally. Its goal to partition nodes of a network into disjoint sets, five different perspectives utilized in network-centric community detection. First is the clustering supported vertex similarity using Jaccard similarity and Cosine similarity. Second is Latent space models supported k-means clustering. Third is the block model approximation based on exchangeable graph models. Fourth is spectral clustering are using minimum cut problem that the number of edges between the two sets is reduced. The fifth is modularity maximization by measures the strength of a community partition by taking into consideration the degree distribution.

✓ Hierarchy centric community detection

Hierarchy centric community detection Aims to build a hierarchical structure of communities supported network topology to permit the analysis of a network at different resolutions two representative methods. First divisive hierarchical clustering (top-down) and the second is agglomerative hierarchical clustering (bottom-up). The strength of a tie is measured by edge betweenness that is the number of shortest paths that pass along with the edge.

For detecting communities the proposed model uses hierarchy centric community detection with topic-driven approach. Here by using LDA topic-model which returned user-topic distributions and by using Jensen-Shannon Divergence algorithm the topic distances between two users have been calculated. Then using the calculated distances, community graph has been constructed. Used an adapted approach of Girvan-Newman which is based on divisive classification. Girvan-Newman Algorithm is as follows.

1. Start all nodes as only one community
2. Calculate betweenness scores for all edges
3. Find edge with highest score and remove it from network
4. Recalculate betweenness for all remaining edges
5. Remove until get communities

3.4. Collaborative Filtering Module

Item-based collaborative filtering (IBCF) was launched by Amazon.com in 1998, which dramatically improved the scalability of recommender systems to cater for millions of customers and millions of items. Prior to the launch of IBCF, there had been many systems of user-based collaborative filtering (UBCF) developed in the academia and the industry, which had the issues of huge computational cost and limited scalability, but since the IBCF algorithm was published in IEEE Internet Computing in 2003, it has been widely adopted across all the Web giants, including YouTube, Netflix, and lots of others.

This article will bring a toy example to explain how UBCF and IBCF work and why internet giants prefer IBCF to UBCF.

✓ User-Based Collaborative Filtering

- Step 1: Calculate the similarity between Alex and all other users

The calculation for the similarity between Alex and Bob can be derived from Formula 1 by putting the corresponding values into the formula as follows: $\text{sim}(\text{Alex}, \text{Bob}) = (4 * 5 + 2 * 3 + 4 * 3) / [\text{sqrt}(4^2 + 2^2 + 4^2) * \text{sqrt}(5^2 + 3^2 + 3^2)] = 0.97$. The similarity value between Alex and Tom can be obtained by following the same way, which is 1.

- Step 2: Predict the ratings of movies that are rated by Alex

In this example, Formula 2-(b) is used as the prediction function. First of all, k value needs to be calculated by injecting the similarity values calculated at Step 1, which is $k = 1/(0.97+1) = 0.51$. Now, the movie Thor unrated by Alex can be worked out by the following calculation: $R(\text{Alex}, \text{Thor}) = k * [\text{sim}(\text{Alex}, \text{Bob}) * R(\text{Bob}, \text{Thor}) + \text{sim}(\text{Alex}, \text{Tom}) * R(\text{Tom}, \text{Thor})] = 0.51 * (0.97 * 4 + 1 * 4) = 4.02$. The final table with the ratings on all movies from Alex is shown in Table 2.

- Step 3: Select top-2 rated movies

Since the ratings of movies that are not rated by Alex have been predicted, it is straightforward to find the top-2 movies, which are Spider-man and Thor.

✓ Item-Based Collaborative Filtering

- Step 1: transpose the user-item matrix to the item-user matrix

As the item similarity is required by IBCF, the item-user matrix shown in Table 3, transposed from the corresponding user-item matrix, makes it more clear by viewing each row as an item vector during the similarity calculation.

- Step 2: Calculate the similarity between any two items and fill up the item-item similarity matrix

First of all, an example of calculating the item similarity between Avenger and Star wars is demonstrated here. According to Formula 3, the specific calculation of the similarity between Avenger and Star wars is as follows: $\text{sim}(\text{Avengers}, \text{Star wars}) = (4 * 2 + 5 * 3) / [\text{sqrt}(4^2 + 5^2) * \text{sqrt}(2^2 + 3^2)] = 0.99624059$. By following the similar way, the item-item similarity matrix can be filled as Table 4, where 0 means the similarity between the two movies cannot be calculated due to data sparsity.

- Step 3: Predict the ratings of movies that are rated by Alex

After successfully building the item-item similarity matrix, the calculation of the rating of movies that are not rated by Alex can be done by injecting the values to Formula 4: $R(\text{Alex}, \text{Thor}) = (\text{sim}(\text{Thor}, \text{Avengers}) * R(\text{Alex}, \text{Avengers}) + \text{sim}(\text{Thor}, \text{Iron man}) * R(\text{Alex}, \text{Iron man})) / (\text{sim}(\text{Thor}, \text{Avengers}) + \text{sim}(\text{Thor}, \text{Iron man})) = 4$.

- Step 4: Select top-2 rated movies for Alex

The final predicted rating of the movie Thor rated by Alex is listed in Table 5 along with the other ratings made by Alex. It is obvious that three movies tie for second place, so the top-2 rated movies by Alex is comprised of Spider-man and one of the three tied movies—Avengers, Thor and Iron man.

- ✓ Computational Cost Comparison Item based vs User based Collaborative Filtering

The computational cost for UBCF in the worst case is $O(NM)$ because it requires examining N customers and up to M items for each customer; However, due to the sparsity of the user-item matrix, the actual computational cost would be close to $O(N + M)$ because for most customers, they only rated a small number of movies which results in a computational cost of $O(N)$, and for a handful customers who rated a significant amount of movies, the computational cost is close to $O(M)$.

In terms of the computational cost for IBCF, there are two parts—building the item-item similarity matrix and predicting the ratings. For building the item-item matrix, $O(N^2M)$ is required in the worst Case, and $O(NM)$ is the computational cost in reality due to the

sparsity in the user-item matrix. In regard to the prediction, the computational cost only depends on the movies that the user has rated, which is usually very little.

By comparing the computational cost of these two methods, it seems that IBCF requires more expensive computational cost, but building the item-item similarity matrix are calculated offline and the online prediction needs very little computational cost; However, for UBCF, there is no offline calculation, so all of the computational cost is online, which turns out that the predictions are literally very slow even for middle-size datasets due to the heavy online computational cost.

- Why not move online computation offline for User-Based Collaborative Filtering

On most of the websites, e.g. Amazon.com, the number of users is much larger than the number of movies. As shown in Table 6, there are seven users and two movies. When the rating of Avenger by Alex changed from 1 to 4, the similarity between Alex with other users are affected drastically as one of the two values in the row/user vector has been changed; while, since the change only takes place in one of the seven values in the column/item vector, it has little impact on the similarity value between Avenger and Star wars. As a result, calculating the similarity between users cannot be moved offline.

- Sparsity issue for User-Based Collaborative Filtering

As most of the users on a website only rated a few of the movies, the data usually are sparse, which could be similar to the user-item matrix in Table 7. Taking the prediction of Bob's rating on Spider-man as an example, IBCF can be easily applied on this task as the item-item similarity can be calculated between Spider-man with any other movies; However, it is hard to achieve an accurate prediction by UBCF because the only similarity value is possible to be calculated between Alex and Bob, and it is impossible for UBCF to produce a decent prediction by using only one similar user's rating.

In proposed system Item based collaborative filtering (3.1) used as the personalization mechanism. User search history in a particular community (based on communities

identified by above community detection module) and according to the frequency of any article viewed or downloaded, article would be given a preference value. These inferred preference values are stored with the search results and they are used to derive a final composite score, on which ultimate search results are based on.

$$P_{ui} = \frac{\sum_{j \in S(i,u)} s_{ij} r_{uj}}{\sum_{j \in S(i,u)} |s_{ij}|} \tag{3.1}$$

3.5. Search Optimization Module

This module is responsible for integrating above modules to search engine, build indexing, rank files, execute the query and find best search results (Figure 3.2). Lucene-lda is used to internalize the topics and topic memberships while building the index and executing the queries. Here used Payloads to cleverly encode the topics in each document at index time. We ignore the actual relevancy returned by Lucene, and instead use the contents of the Payload to compute the relevancy ourselves, and re-rank the results.

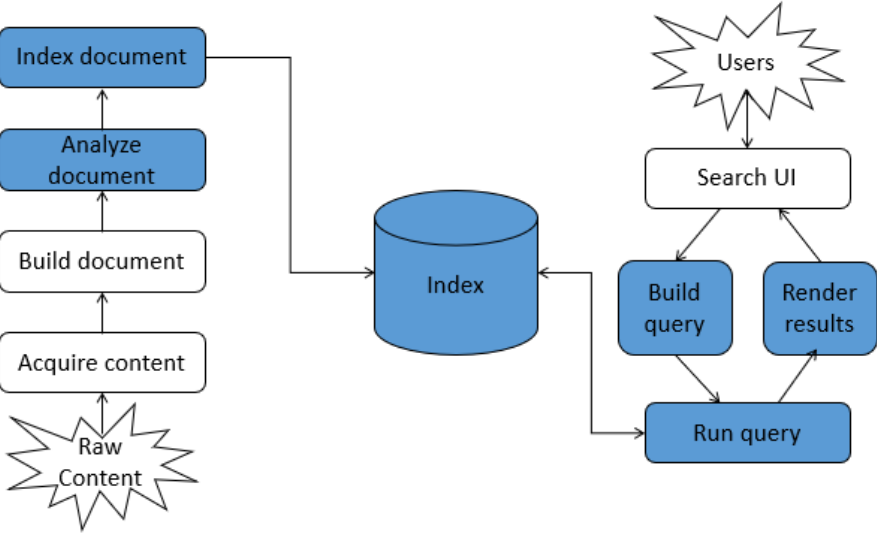


Figure 3.2 - Lucene in a search system

3.6. Summary

Chapter 3 described about the technologies used in discovering hidden topics using topic modeling, community detection and collaborative filtering techniques. Chapter 4 will discuss about the approach taken.

4. APPROACH

4.1. Introduction

Chapter 3 presented a broad overview of technology that have been used for solving semantic information retrieval system. In this chapter we provide our novel approach to semantic information retrieval with the use of semantic web and collaborative filtering. As such we have structure the chapter with subsections namely hypothesis, input, output and overall features. In the description, the section of the process gives over all functionality of the system together with the relevant technologies.

4.2. Hypothesis

Proper identification of topics in a corpus, identification of shared interests in a community based on topic-driven approach and recommending items based on item-based collaborative filtering will improve providing more relevant search results to user in a semantic information retrieval platform in a web application.

Proposed solution is to provide a semantic information retrieval platform for internal searching of a software application. Information retrieval would be based on semantics of data, based on topic-based community detection, based on applying personalization and collaborative filtering on top of detected communities for providing more meaningful search results to user. To provide semantically rich search results, application will look into latent Dirichlet allocation and topic-driven community detection methods integrating with collaborative filtering.

The system contains five modules called

1. UI module which provide user interface for uploading documents and entering search query. Here user can create documents and folders also. User can view, edit, delete uploaded files. User can configure the settings related to their company.

2. Discover topics Module which discover hidden topics in user uploaded documents. First the documents are preprocessed and then train LDA model and by using LDA model do topic extraction. These extracted topics are used to define communities and indexing search engine.

3. Community extraction module which is detecting communities based on topic-driven approach and community detection algorithms. Here by using LDA topic-model which returned user-topic distributions and by using Jensen-Shannon Divergence algorithm, the topic distances between two users are calculated. Then using the calculated distances, community graph has been constructed. Then to construct use communities used an adapted approach of Girvan-Newman which is based on divisive classification.

4. Item based collaborative filtering Module which uses the detected communities and user history to find more relevant articles. Item based collaborative filtering consider about the user search history in a particular community (based on communities identified by above community detection module) and according to the frequency of any article viewed or downloaded, article would be given a preference value. These inferred preference values are stored with the search results and they are used to derive a final composite score, on which ultimate search results are based on.

5. Search optimization Module which is responsible for integrating above modules to search engine, build indexing, rank files, execute the query and find best search results. Lucene is used to internalize the topics and topic memberships while building the index and executing the queries. Here used Payloads to cleverly encode the topics in each document at index time. When user has entered the query, determine which topics are in the query, based on the terms in the query. Then create a Payload query based on these topics. Lucene will then find all documents that contain these topics. We ignore the actual relevancy returned by Lucene, and instead use the contents of the Payload to compute the relevancy ourselves, and re-rank the results.

4.3. Process

Search engines or localized software systems developed for information searching, play an important role in knowledge discovery. Proliferation of data in the web and social media has posed significant challenges in finding relevant information efficiently even using those search engines or other software systems. Moreover, those systems or engines tend to collect massive number of data, which could be useful for humans in various ways but overlook the meaning of the search phrases, hence generate irrelevant search results. A unit level searching i.e. searching information within a website or page is also not effective as they follow exact keyword matching techniques and ignore the semantic level matching of search phrases. In order to address those deficiencies, this research proposes a hybrid approach which use the semantics of data, community preferences as well as collaborative filtering techniques for semantic information retrieval. More specifically, Topic modeling based on Latent Dirichlet Allocation together with topic-driven based community detection methods are applied for identifying personalized search results and hence improve the relatedness of the research results. Based on the proposed hybrid approach a framework for semantic search that can easily be integrated to a software application has been implemented. The evaluation results confirm the effectiveness of search results which outperform benchmark approaches that follow traditional keyword search algorithms.

The process of the system behaves like finding the needle inside hay. The system uses topic modeling to detect most relevant topics for given corpus, uses topic-driven community detection algorithm to detect communities and provides most relevant search results based on community interests and item-based collaborative filtering.

1. User login and authentication
2. User upload files
3. Train the model with given corpus
4. Discover communities
5. Analyze user uploaded files and community interests

6. Detect most relevant files for user given query

The top-level architecture of our proposed MAS system is shown figure 4.1

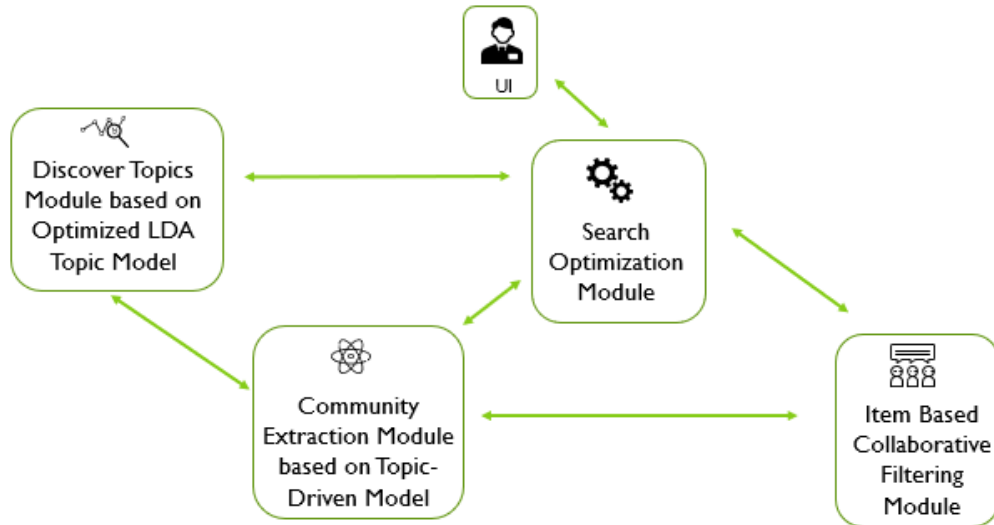


Figure 4.1 - Design diagram (Top level architecture)

4.4. Input and Output

Files uploaded by the users, user identity (unique user-id), search query entered by user are the main inputs to the system. Recommended search results for user-entered query with suggested document list are the main output of the system.

User has to sign-up first and after sign in user can upload files, create files, edit files. Based on users context he will be assigned to communities dynamically and when user search for particular thing, results will display based on user community preferences.

4.5. Users

The people who are using the internal web applications can search their queries using this application. Also any person can use this system and can search for particular article.

4.6. Features

Provide semantic search results based on semantic meaning, recommend files based on detected hidden communities are the key features of this system. The system is an online application which can access from any location. User can configure their own settings and can use the system.

4.7. Summary

Chapter 4 described about the approach used in semantic metadata extraction and collaborative filtering techniques. Chapter 5 will discuss about the design of the internal web application including information retrieval system.

5. DESIGN

5.1. Introduction

Chapter 4 discuss about the approach taken for the development of research. This chapter presents top level architecture of semantic information retrieval of internal web application. It contains four different modules which altogether perform the semantic searching. The file server would be located in a separate location. The framework is responsible of handling searching functionality.

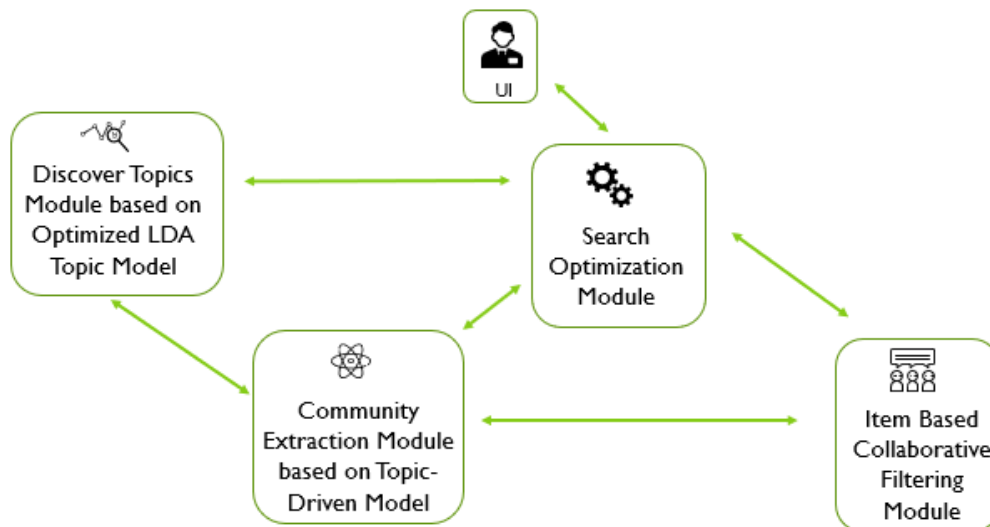


Figure 5.1 - High level architecture of semantic information retrieval

Semantic information retrieval platform has five main modules (Figure 5.1).

Proposed solution is to provide a semantic information retrieval platform for internal searching of a web application. Information retrieval would be based on identifying hidden topics of the articles, based on identifying hidden communities using topic-driven approach and based on applying personalization and collaborative filtering for providing more meaningful search results to user. To provide semantically rich search results,

application will look into latent Dirichlet allocation and Jensen-Shannon-Divergence for community extraction integrating with collaborative filtering.

5.2. Discover Topics Module based on Optimized LDA Model

Topic modeling is a model which uses generative statistical models and analyze the words in a collection of documents to discover the hidden semantic structure in these documents. The proposed system use, is the well-known statistical topic model, Latent Dirichlet Allocation (LDA) [15,16]. In LDA model (Figure 5.2), the hidden semantic structure includes:

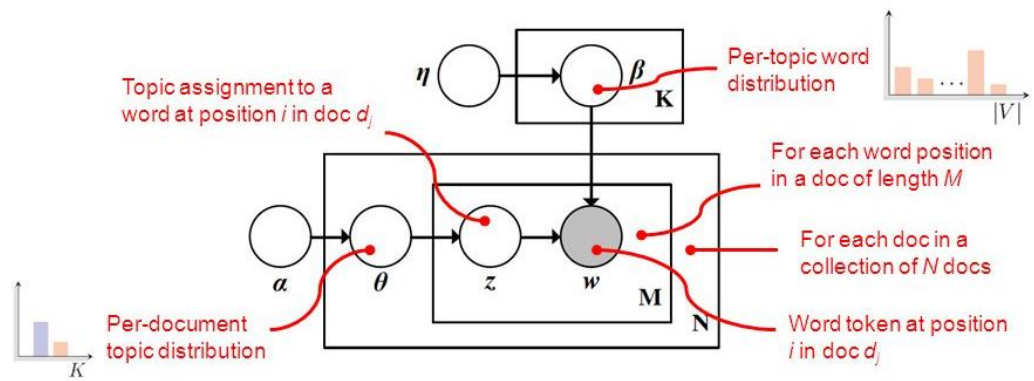


Figure 5.2 - LDA model

1. a list of topics, i.e., the topics occurring in the collection of documents
2. the per-topic word distribution ϕ for a specific topic, i.e., the list of words in a topic and the probability of a word appearing in a topic
3. the per-document topic distribution θ for a specific document, i.e., the probability that a document covers each topic,
4. the per-document per-word topic word assignments z , i.e., the topic for a word in a document.

In other words, LDA represents documents as random mixtures over latent topics that generate words with certain probabilities.

Assuming this generative model for a collection of documents, the central computing problem for LDA is to inspect the observed collection to find the hidden topic structure that is most likely to generate the collection, that is, to figure out the latent variables ϕ , θ , and z . In addition, the above LDA model is conditioned on three parameters, α and β , and k , where α is the parameter of the Dirichlet prior on the per-document topic distribution, β is the parameter of the Dirichlet prior on the per-topic word distribution, and k is the number of topics covered by the collection of documents.

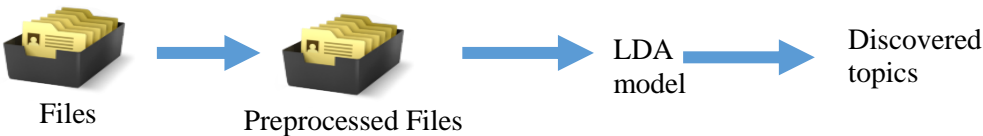


Figure 5.3 - Process of discovering hidden topics from the set of articles

Figure 5.3 illustrate the process of discovering hidden topics from the set of articles. The application will iterate through all the documents in document server and preprocess the data by removing unnecessary words. Then the preprocessed articles will be sent to LDA model for training purpose. The LDA model will discover the hidden topics of the articles.

5.3. Community Extraction Module based on Topic-Driven Model

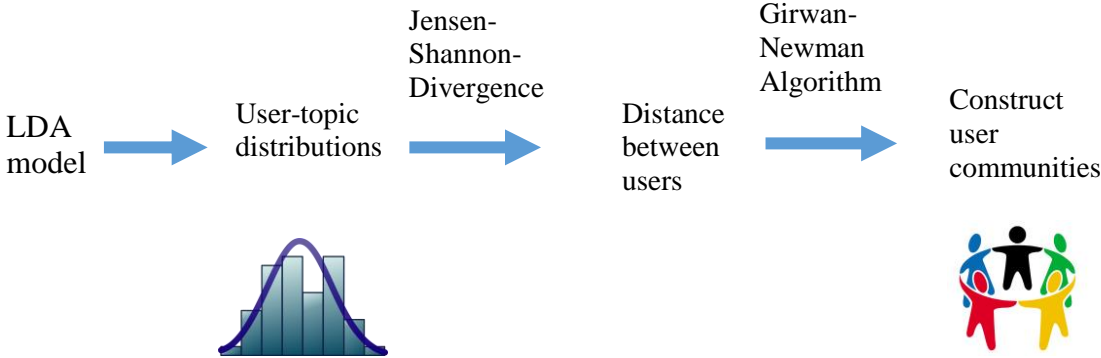


Figure 5.4 – Topic-driven community detection module

Figure 5.4 describes the process of topic-driven community detection module. Following steps have been followed in this process.

- Calculate distance between users

The researchers, in [23], define the distance between user i and user j as the Jensen-Shannon Divergence between the topics distributions on users presented by the following (5.1).

$$dist_T(i, j) = \sqrt{2 * D_{JS}(i, j)} \quad (5.1)$$

where $D_{JS}(i, j)$: the Jensen-Shannon Divergence between the two topic distributions DT_i and DT_j . It is defined as (5.2).

$$D_{JS}(i, j) = \frac{1}{2}(D_{KL}(DT_i||M) + D_{KL}(DT_j||M)) \quad (5.2)$$

where M : the average of the two probability distributions. D_{KL} : the Kullback-Leibler Divergence which defines the divergence from distribution Q to distribution P as (5.3).

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} \quad (5.3)$$

In the first measure (domains), we calculate the distance between users as the Jensen-Shannon Divergence between domains distributions over users as in formula 1 and 2. – The second measure (topics-domains) combines the two previous measures (Topics, Domains). This new measure allows decreasing the distance between users who do not treat only same topics but also same domains. The distance between users in this measure is computed by using (5.4).

$$dist_{TD}(i, j) = \sqrt{2 * D_{JS}(i, j)} \quad (5.4)$$

- Construct graph

Here nodes and edges represent users and topic distance between users respectively. In the topic graph, we create an edge from the user i to the user j , if the user j is the closest to the user i for the topic k , the weight of this link is calculated as the distance between them for this selected topic k . Moreover, if there is another edge from the user i to the user j for another topic, it is enough to choose the minimal distance between these two users i and j (Figure 5.5).

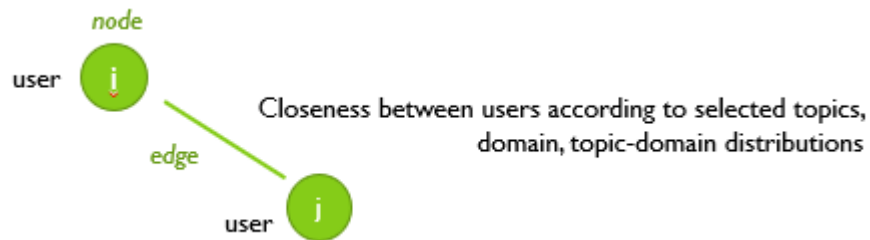


Figure 5.5 - Graph representation

- Construct user communities

Used an adapted approach of Girvan-Newman which is based on divisive classification. Girvan-Newman Algorithm is as follows.

- Start all nodes as only one community
- Calculate betweenness scores for all edges
- Find edge with highest score and remove it from network
- Recalculate betweenness for all remaining edges
- Remove until get communities

Anthropologist Dunbar [24] suggests that the size of communities with strong ties in both traditional social networks and Internet-based social networks should be limited to 150 (called Dunbar's number) because of human's cognitive constraints and time constraints. Large communities of size over 150 people contain weak connections

among their members and are therefore not stable. Therefore, the community size was limited to 150 users.

5.4. Item-based Collaborative Filtering Module

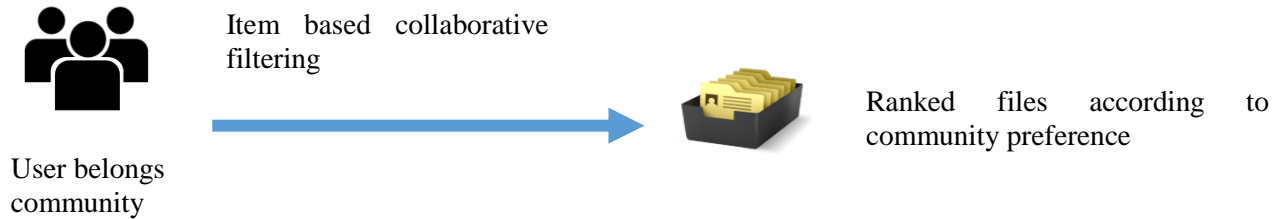


Figure 5.6 – Process of ranking files according to dynamically defined community preferences

Figure 5.6 shows how files have been ranked according to users' interest in a community. Here item based collaborative filtering is used.

Item based collaborative filtering consider about the user search history in a particular community (based on communities identified by above community detection module) and according to the frequency of any article viewed or downloaded, article would be given a preference value. These inferred preference values are stored with the search results and they are used to derive a final composite score, on which ultimate search results are based on.

Here pairwise similarities of the columns of the rating matrix using some similarity measure is computed and store top 20 to 50 most similar items per item in the item-similarity matrix. Prediction: use a weighted sum over all items similar to the unknown item that have been rated by the current user (5.5). The output of this module is list of recommended articles for particular user

$$P_{ui} = \frac{\sum_{j \in S(i,u)} s_{ij} r_{uj}}{\sum_{j \in S(i,u)} |s_{ij}|} \quad (5.5)$$

5.5. Search Optimization Module

This module is responsible for integrating above modules to search engine via build indexing, rank files, execute the query and find best search results (Figure 5.7).

Lucene is used to internalize the topics and topic memberships while building the index and executing the queries [25]. List of terms in the corpus (term list), matrix that specifies the membership of each word in each topic. (topic-word distribution), matrix that lists the original file names that LDA was executed on and matrix that specifies the topic membership of each file in each topic. (document-topic distribution) are the input files to system.

Here used Payloads to cleverly encode the topics in each document at index time. When user has entered the query, determine which topics are in the query, based on the terms in the query. Then create a Payload query based on these topics. Lucene will then find all documents that contain these topics. We ignore the actual relevancy returned by Lucene, and instead use the contents of the Payload to compute the relevancy ourselves, and re-rank the results.

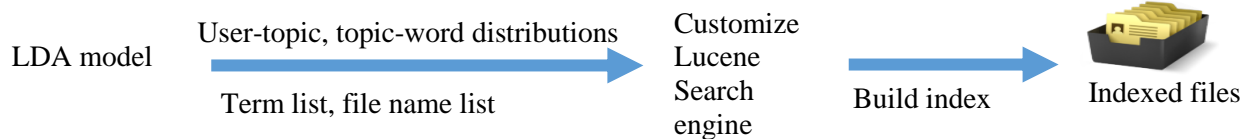


Figure 5.7 – Process of indexing articles

5.6. Summary

Chapter 5 discuss about the design of the semantic information retrieval system. Chapter 6 will discuss about how the implementation of the semantic information retrieval system has done for internal web application.

CHAPTER 6

6. IMPLEMENTATION

6.1. Introduction

In this chapter we are discussing detail about how each module described in chapter 4 has been implemented. Semantic Information Retrieval Platform has been designed to run on any Hardware platform since it is a web application. The overall software has been developed as an ASP .Net web application. However different modules inside the system have been implemented with different technologies such as Python programming language using large number of libraries like Gensim, NLTK, MALLET etc, ANACONDA platform for easy source code maintenance and Lucene has used as Search Engine platform for indexing and integrating other modules.

Following Figure 6.1 describes the entire flow of the system.

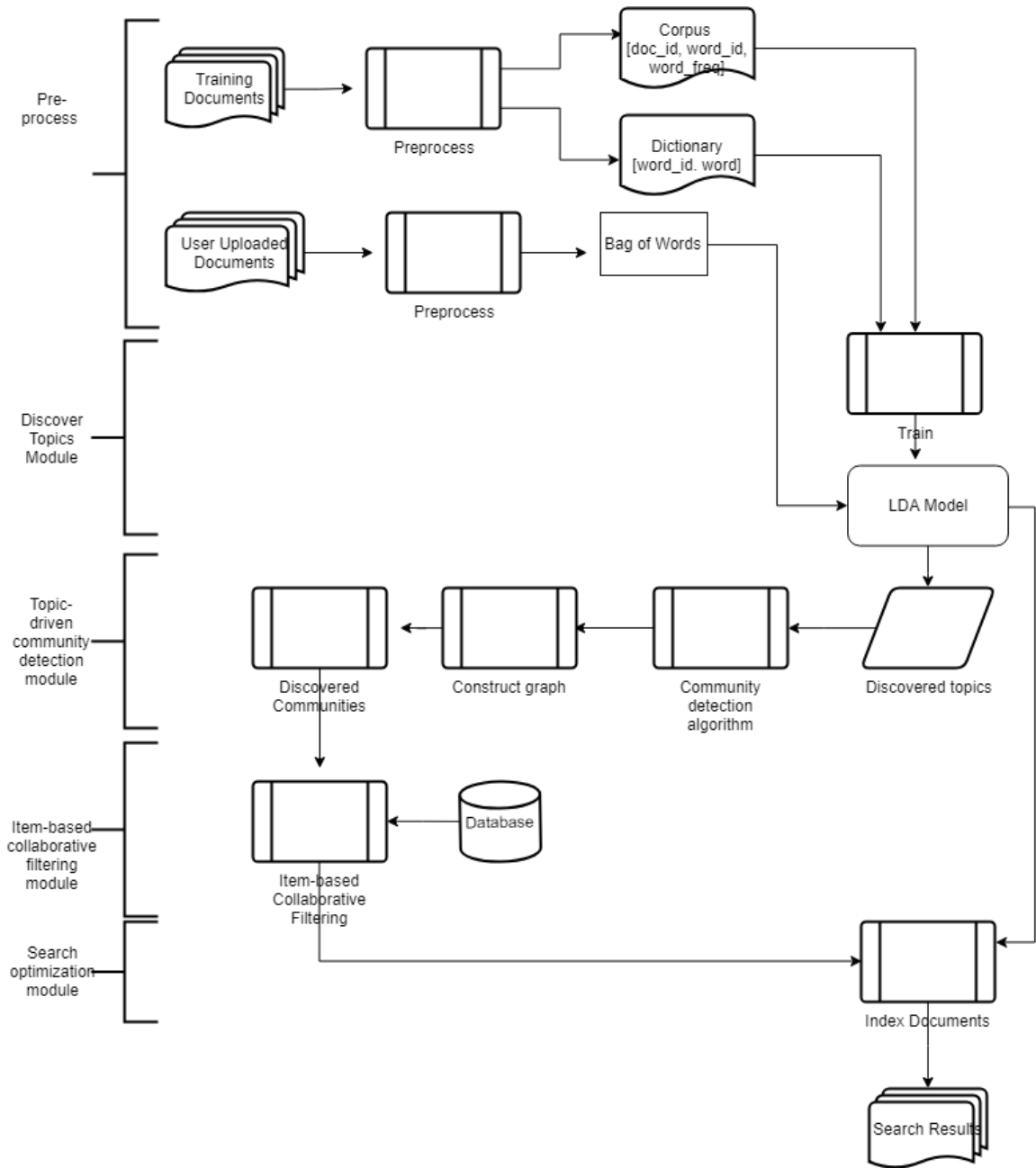


Figure 6.11 - Semantic Information Retrieval System

6.2. Data Preprocessing

The training module uses XML parsing of Wiki Dump with 70,000 articles for training LDA model. Before passing articles to train the model, those Wiki articles are preprocessed by the removal of stop words, URLs, articles, file attachments, XML labels, special characters, digits, spaces, new lines, punctuations etc. It Uses lemmatization for further filtering of necessary data. Lemmatization is nothing but converting a word to its root word. For example: the lemma of the word ‘machines’ is ‘machine’. Likewise, ‘walking’ → ‘walk’, ‘mice’ → ‘mouse’ and so on. Keep only articles with more than 150 characters and it considers English characters only. It produces a corpus as a list of document id, word id, word frequency -dictionary (Figure 6.2) and a list of words with their ids. All the tokens(words) in the dictionary which either have occurred in less than 4 articles or have occurred in more than 40% of the articles are removed from the dictionary. Further removed content neutral words and finally Prepare document-term matrix. For more optimization created Bigram and Trigram Models. The technologies used here is NLTK package comes with Gensim library in Python (Appendix A).

```
words frequency
[(u'state', 10294), (u'one', 9451), (u'unite', 9213), (u'first', 8511),
(u'american', 8383), (u'name', 6933), (u'play', 6043), (u'new', 5701),
(u'bear', 5624), (u'two', 5614), (u'time', 5523), (u'world', 4949)]
ids
[22871, 579, 19641, 3768, 2573, 18650, 19284, 6702, 24598, 17353, 20208,
4284]
```

Figure 6.2 - Word frequency dictionary

6.3. Discover Topics Based on Topic Modeling

The module uses the Latent Dirichlet Allocation (LDA) from Gensim package along with the Mallet's implementation (via Gensim). Mallet has an efficient implementation of the LDA. It is known to run faster and gives better topics segregation. Figure 6.3 demonstrated the inputs and outputs of LDA model. As per the technology, here used Python 2.7 as programming language and Python core packages used - re, gensim, spacy, pyLDAvis, matplotlib, numpy and pandas.

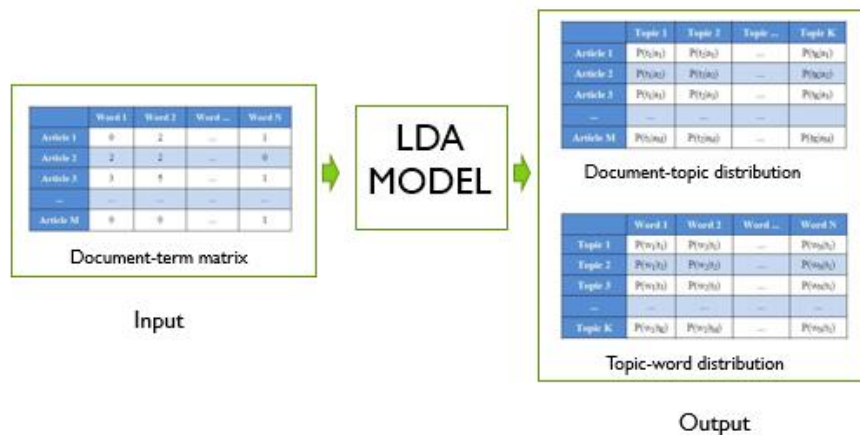


Figure 6.3 - LDA Model

- Optimize LDA model

Computed Model Perplexity and Coherence Score which provides a convenient measure to judge how good a given topic model is. Perplexity lower is the better and coherence higher is the better. Further used MALLET to optimize LDA model. Build many LDA models with different values of number of topics (k) and pick the one that gives the highest coherence value.

```
7.  
8. #Creating the object for LDA model using gensim library & Training LDA  
   model on the document term matrix.  
9. ldamodel = Lda(doc_term_matrix, num_topics=50, id2word = dictionary,  
                 passes=50, iterations=500)
```

```

10.ldafile = open('lda_model_sym_wiki.pkl','wb')
11.cPickle.dump(ldamodel,ldafile)
12.ldafile.close()
13.
14.#Print all the 50 topics
15.for topic in ldamodel.print_topics(num_topics=50, num_words=10):
16.    print topic[0]+1, " ", topic[1],"\n"

```

When build the LDA model, following parameters need to be passed.

```

# Build LDA model
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                             id2word=id2word,
                                             num_topics=50,
                                             random_state=100,
                                             update_every=1,
                                             chunksize=100,
                                             passes=10,
                                             alpha='auto',
                                             per_word_topics=True)

```

- Key factors to obtain good segregation topics
 1. The quality of text processing.
 2. The variety of topics the text talks about.
 3. The choice of topic modeling algorithm.
 4. The number of topics fed to the algorithm.
 5. The algorithms tuning parameters.

6.3. Community Detection Based on Topic-Driven Approach

For the community detection following steps have been followed.

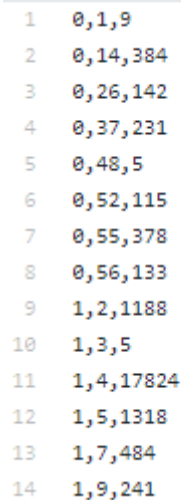
1. Calculate the distance between users.

Here used Jensen-Shannon-Divergence in order to calculate the distances between users according to topic distributions. As per the technology used Python programming language with necessary python libraries (Appendix B)

2. Construct the graph

Here nodes and edges represent users and topic distance between users respectively. In the topic graph, we create an edge from the user i to the user j , if the user j is the closest to the user i for the topic k , the weight of this link is calculated as the distance between them for this selected topic k . Moreover, if there is another edge from the user i to the user j for another topic, it is enough to choose the minimal distance between these two users i and j .

This graph construction was implemented by writing algorithm based on user-topic distributions discovered from LDA model.



```
1 0,1,9
2 0,14,384
3 0,26,142
4 0,37,231
5 0,48,5
6 0,52,115
7 0,55,378
8 0,56,133
9 1,2,1188
10 1,3,5
11 1,4,17824
12 1,5,1318
13 1,7,484
14 1,9,241
```

Figure 6.4 - Sample Graph.txt file content

The weighted graph “graph.txt” (Figure 6.4), it's a CSV file where each line has the following format: u,v,w . Above line specifies that there is an edge between node u and v with positive weight w .

3. Construct communities

Following code describes the community construction once we input above constructed graph structure from the file (Appendix C).

Output of this module is a list of communities with their belonging users. For an example in Figure 6.5 dictates that users 1,2,3 belongs to one community and users 9,10,11 belongs to another community.

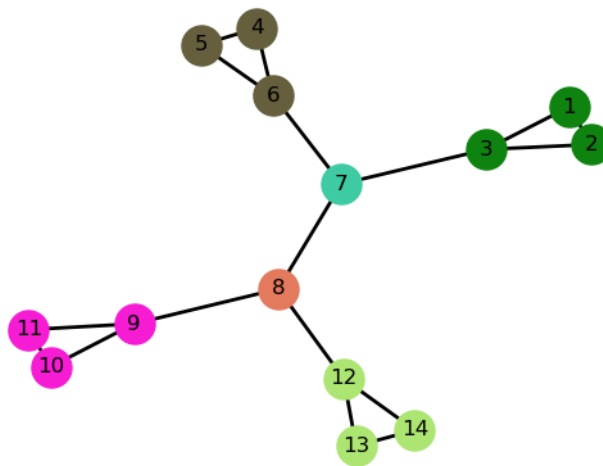


Figure 6.5 - Constructed communities

6.4. Item-Based Collaborative Filtering Module

Item based collaborative filtering consider about the user search history in a particular community (based on communities identified by above module) and according to the frequency of any article viewed or downloaded, article would be ranked. Here the python implementation interacts with the database search history (Appendix D).

The output of this module is list of recommended articles for particular user.

6.5. Search Optimization Module

This module is responsible for integrating above modules to search engine, build indexing, rank files, execute the query and find best search results. Lucene-Lda is used to internalize the topics and topic memberships while building the index and executing the queries.

Specifically, need to specify four files, for each parameter LDA parameter combination

1. vocab.dat: a $V \times 1$ list of terms in the corpus (term list)
2. words.dat: a $K \times V$ matrix (white-space delimited) that specifies the membership of each word in each topic. (topic-word distribution)
3. files.dat: A $D \times 3$ matrix (white-space) that lists the original file names that LDA was executed on. The first and third columns are ignored; the second column should contain the file name. (document name list)
4. theta.dat: A $D \times K$ matrix (white-space) that specifies the topic membership of each file in each topic. (document-topic distribution) –pass only user’s community $D \times K$ only, if search result is empty, then only pass all $D \times K$

V is the number of terms; K is the number of topics; and D is the number of documents. The order of the terms in vocab.dat should match the order in words.dat; the same is true for the filenames in files.dat and theta.dat.

Here used Payloads to cleverly encode the topics in each document at index time. When user has entered the query, determine which topics are in the query, based on the terms in the query. Then create a Payload query based on these topics. Lucene will then find all documents that contain these topics. We ignore the actual relevancy returned by Lucene, and instead use the contents of the Payload to compute the relevancy ourselves, and re-rank the results. In order to provide more better results, recommended files returned from item-based collaborative filtering module is integrated when provide search results to user. Following C# code snippet describes how LDA model has integrated to Lucene search engine (Appendix E).

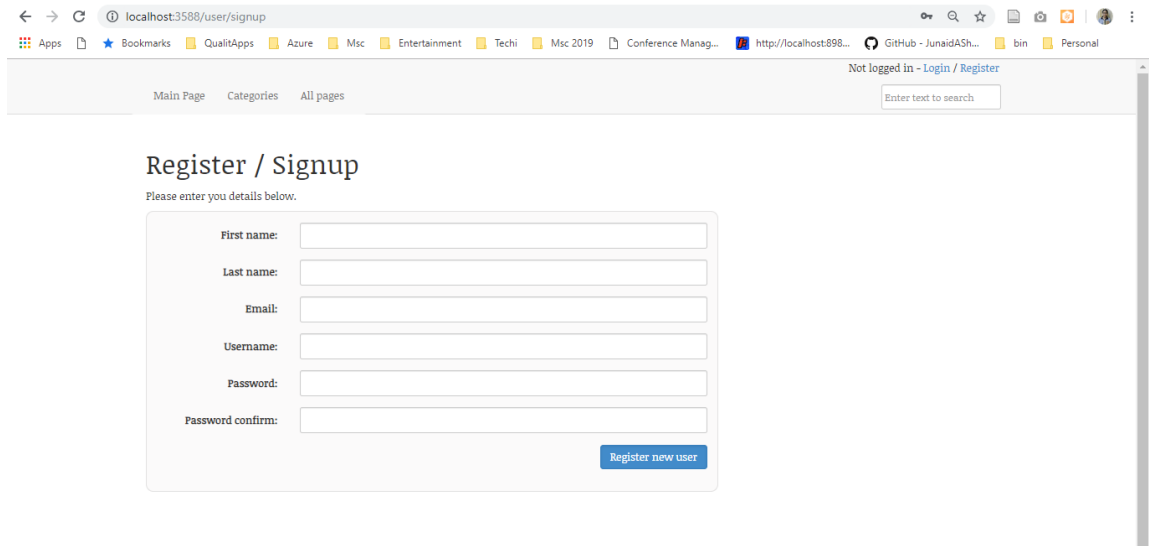


Figure 6.6 - User Sign-up page

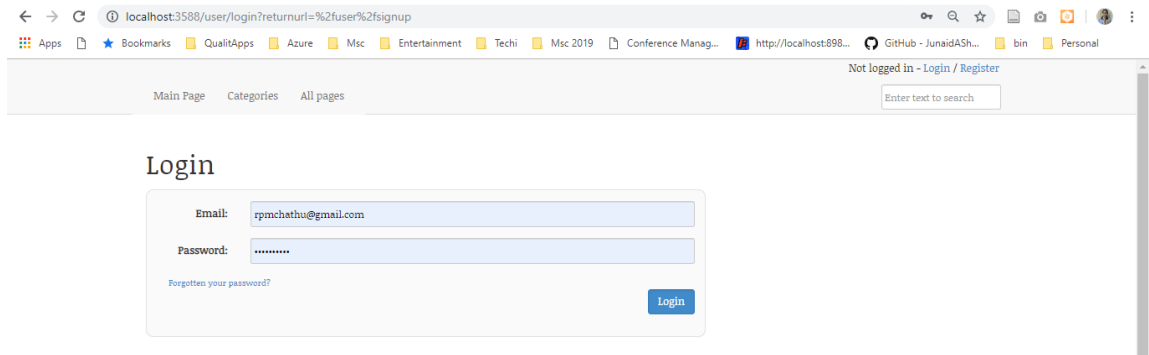


Figure 6.7 - User sign-in page

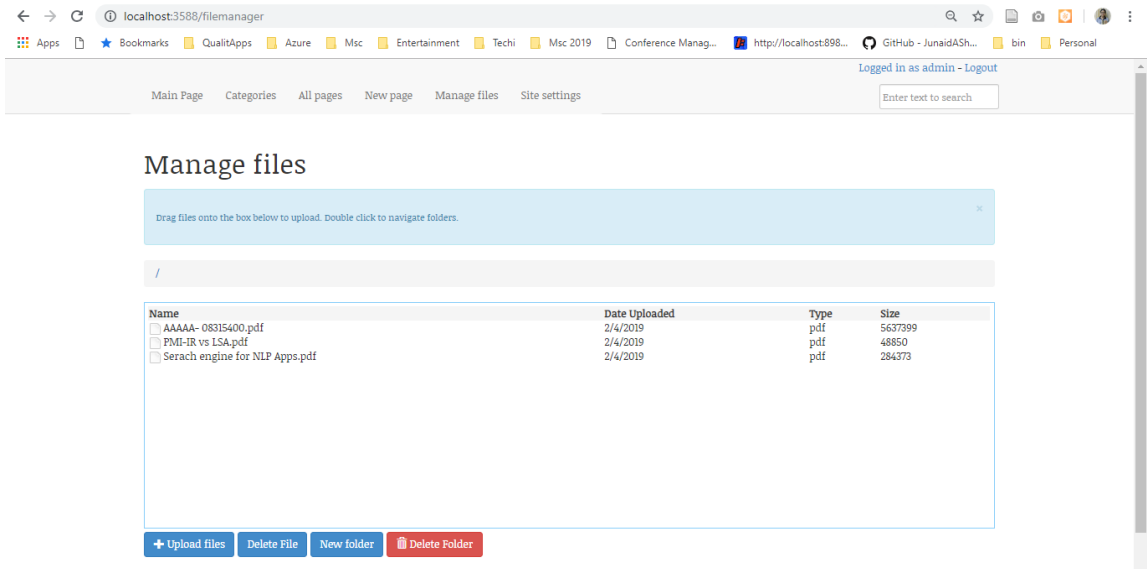


Figure 6.7 - List of user uploaded files and user can edit uploaded files

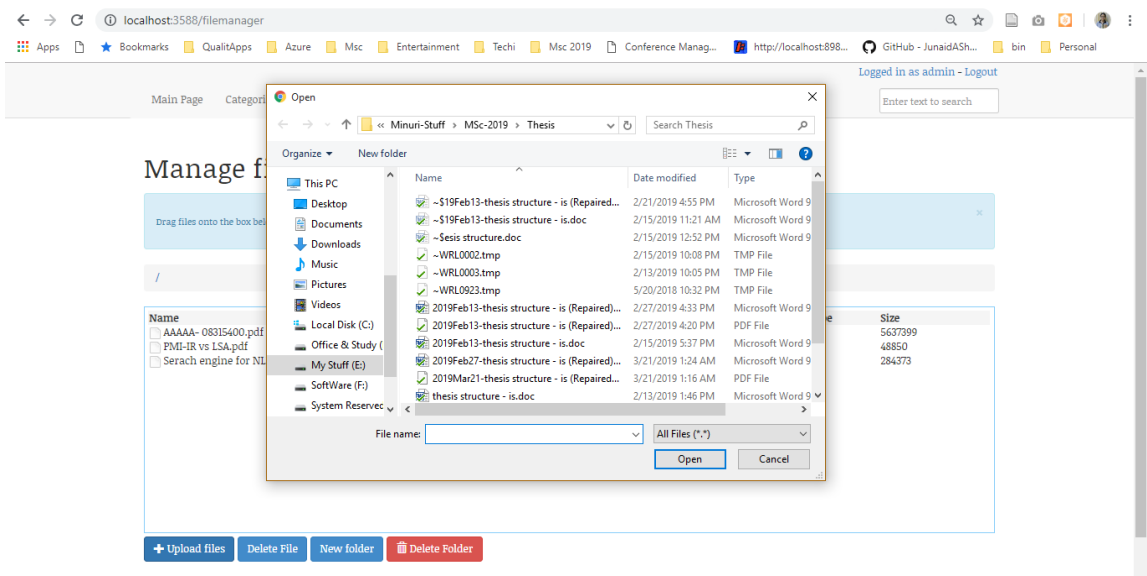


Figure 6.8 - File up loader

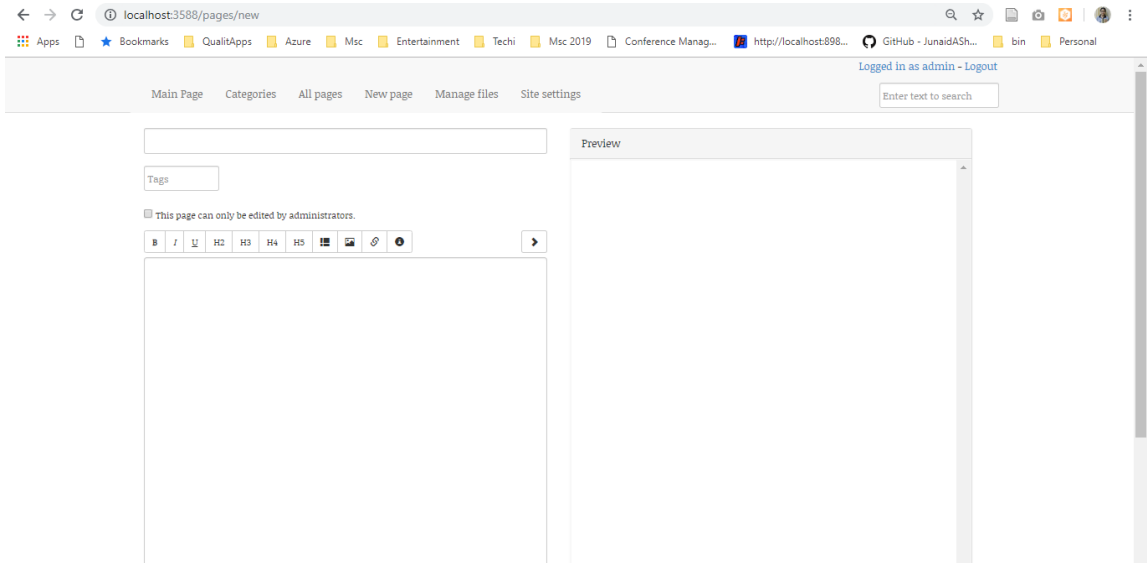


Figure 6.9 - Create new file

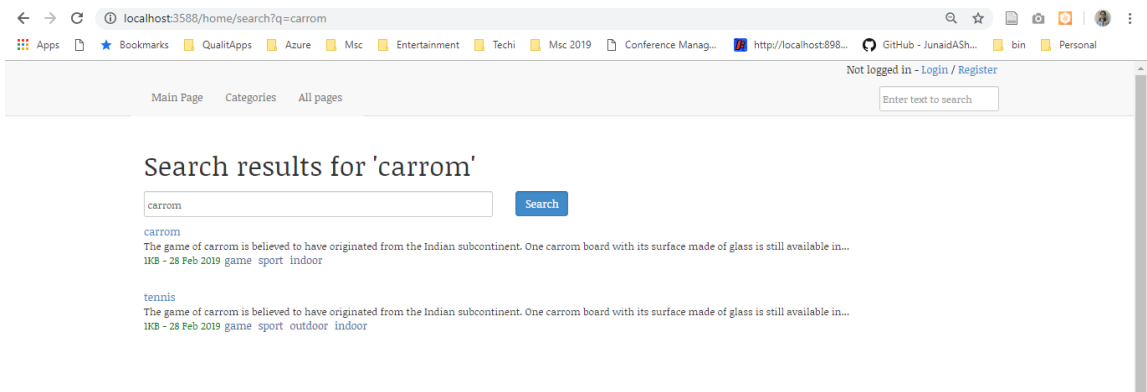


Figure 6.10 - Search results for text “carrom”

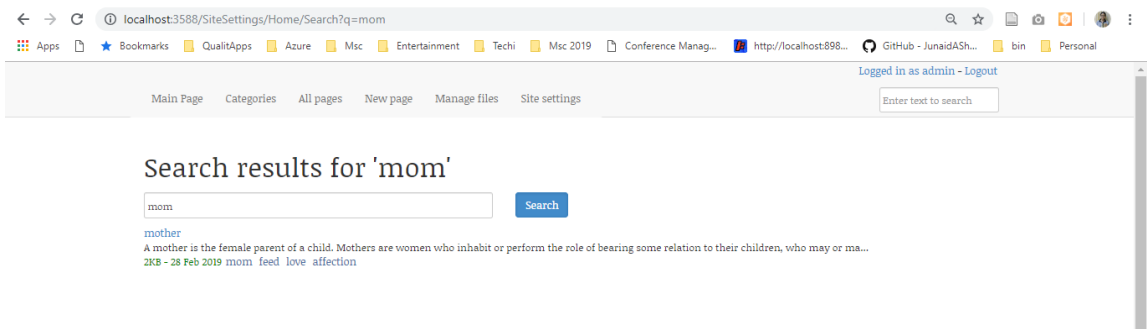


Figure 6.11 - Search results for text “mom”

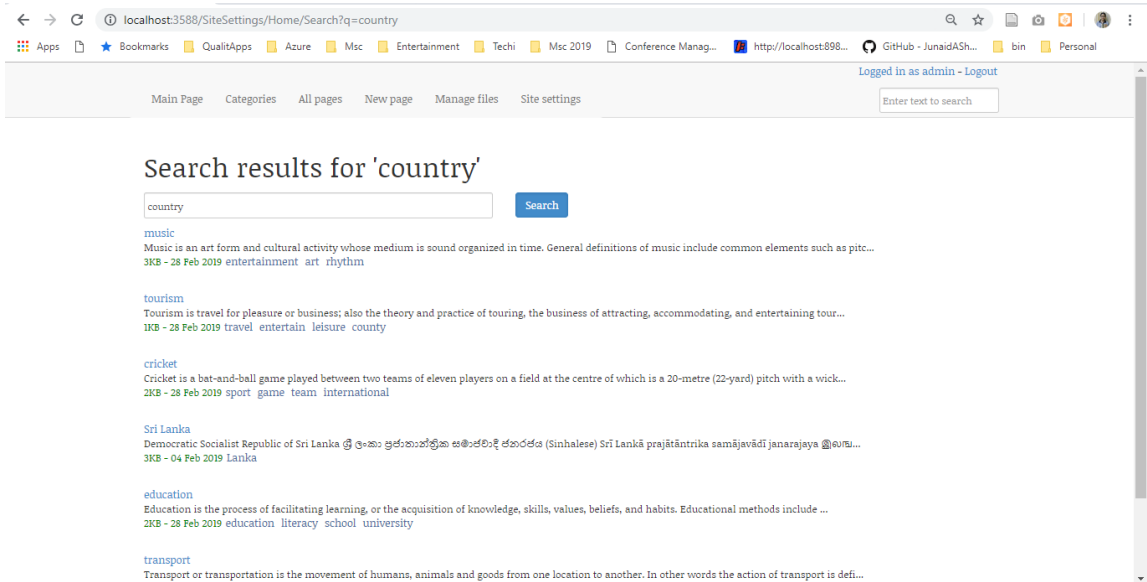


Figure 6.12 - Search results for text “country”

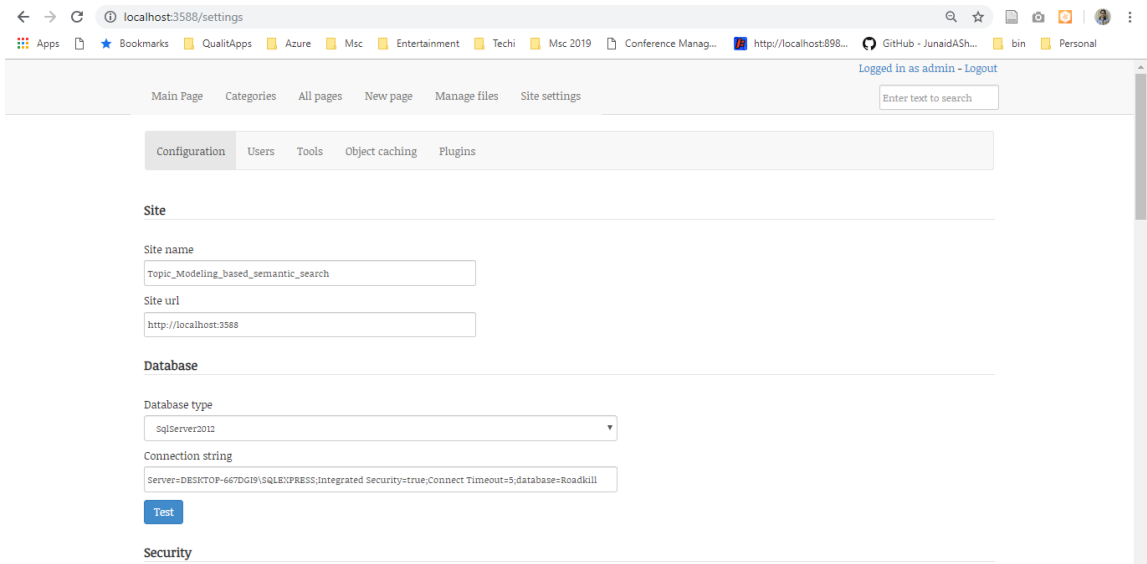


Figure 6.13 - Configuration set-up page

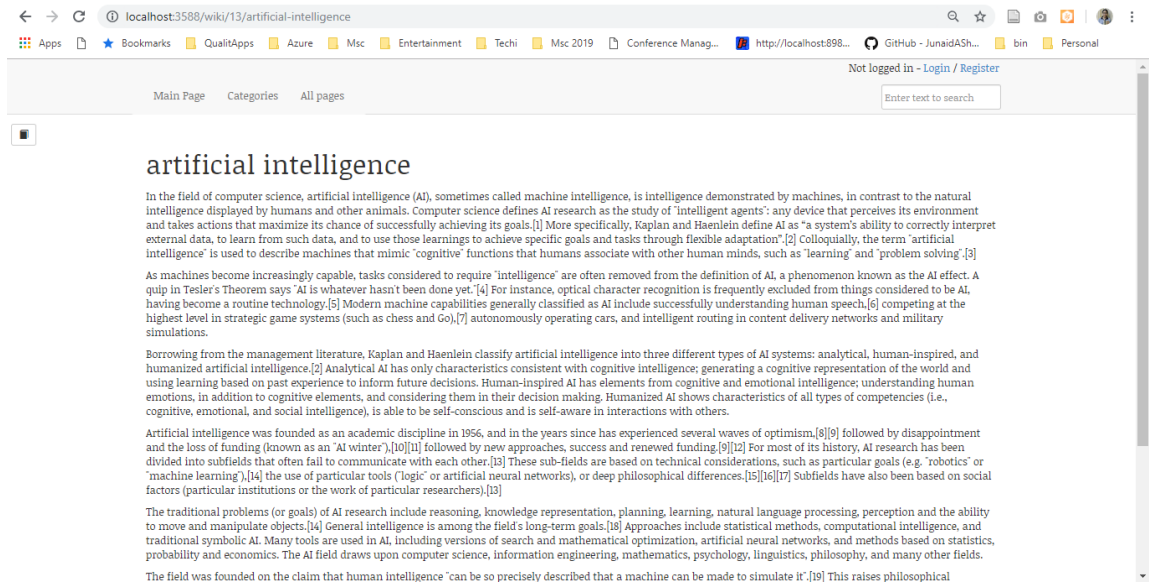


Figure 6.14 - Uploaded file content

6.6. Summary

Chapter 6 discuss about the implementation of the semantic information retrieval system explaining in detail how each module has implemented. Chapter 7 will discuss evaluation of the semantic information retrieval system has done for internal web application.

CHAPTER 7

7. EVALUATION

7.1. Introduction

This section presents the observations on the discovered topic accuracy and topic similarities of users in dynamically defined communities. Here explained the Experimental Setup, Participants, Test Cases, Testing strategies. In detail explain about the collected training corpus, quality of trained LDA model and further more compare the topic similarity among users in an actual group versus, users in a dynamically detected community. Hence this chapter presents the quality of entire proposed model for obtaining semantic information with community interests.

7.2. Experimental Setup

Experiment done in personal computers, which installed python libraries.

7.3. Participants

The participants are cooperative 20 web application users which are employees of QualitApps Asia Pvt Ltd. Experiment done in personal computers, which installed python libraries. Users have given their own user name and passwords to login to the system.

7.4. Test Cases

Input - Group of users has to note down what topics they suggest for given set of documents. User has to save set of files inside the user specific folder. User has to define his particular interested area. User enters query to search for articles.

Output – Topics identified for given set of articles by the system. Constructed user communities with user belonginess. Recommended articles for user entered query based on community interests.

7.5. Testing Strategies

Install the application to run on Linux, Apple and Windows environment.

7.6. Training Data

The corpus used to train an LDA model is collection of articles in English Wikipedia which was downloaded from [26] in September 2018. The collection contains over 5 million articles. Articles are preprocessed by removal of unnecessary words. stop words, URLs, articles, file attachments, XML labels, special characters, digits, spaces, new lines, punctuations etc. It Uses lemmatization for further filtering of necessary data. Lemmatization is nothing but converting a word to its root word. For example: the lemma of the word ‘machines’ is ‘machine’. Likewise, ‘walking’ → ‘walk’, ‘mice’ → ‘mouse’ and so on. Keep only articles with more than 150 characters and it considers English characters only. It produces a corpus as a list of document id, word id, word frequency -dictionary and a list of words with their ids. All the tokens(words) in the dictionary which either have occurred in less than 4 articles or have occurred in more than 40% of the articles are removed from the dictionary.

7.7. LDA Model Performance

LDA model is trained by setting the number of topics to 10, 20, and 40 respectively. Fig. 7.1 illustrates computed coherence score when number of topics(K) equals to 10, 20 and 40. In LDA model higher the coherence score means the trained topic model is more accurate. It can be justified when compare with Table 7.1. Table 7.1 interprets the actual topics defined for given articles for different K values along with human prediction. When comparing those two results we can interpret that the best LDA model creates when K=20 since it is closer to topics defined by human.

Table 7.1 - Evaluate the model based on different topic numbers

A	K = 10	K = 20	K = 40	Human
1	Government	War	Game	War
2	People	Education_Research	Literature_or_Study	Education
3	People	Movie_TV_SHow	Music	Music
4	Award_Ceremony	Games	People	Sports
5	Space_Astronomes	Weather_or_Sport	Natural Disaster	Weather
6	Unknown	Engineering	Engineering	Computer
7	LandScapes	Nature	Natural Species	Biology
8	Governing	Finance	Unknown	Business
9	Country	Computer	Software_Computer	Engineering
10	Relationships	Food_Meals	unknown	Food

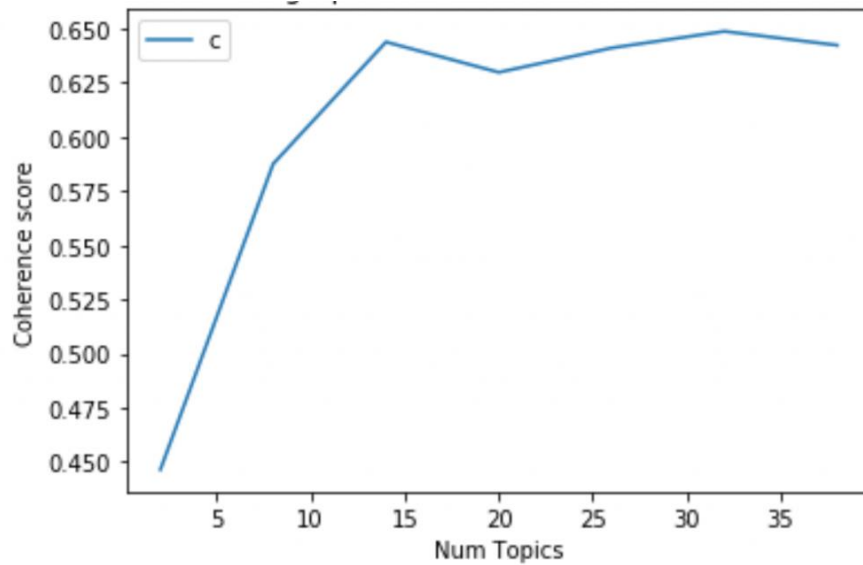


Figure 7.1 - Choosing optimal model with coherence scores

Figures 7.2, 7.3 illustrates the interpretations have done over the trained LDA model.

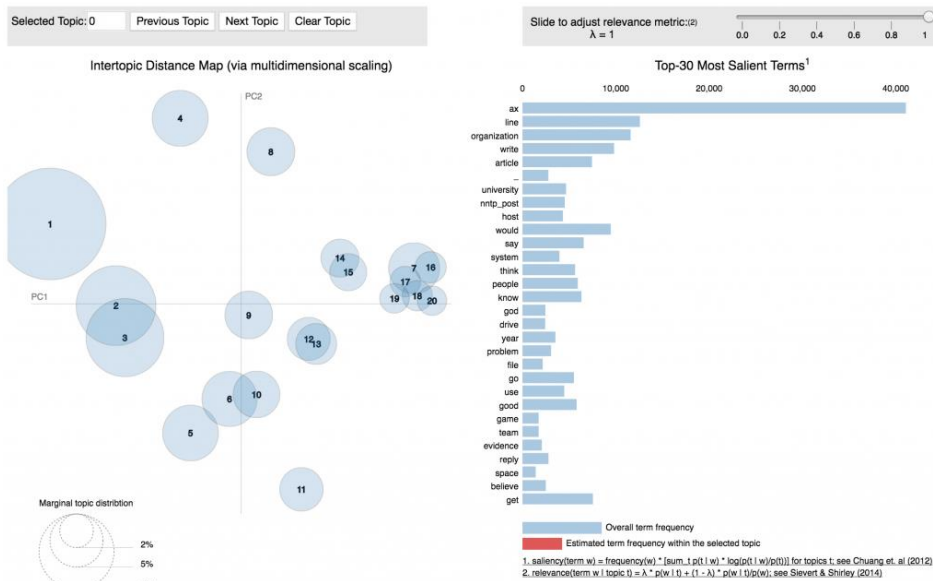


Figure 7.2 - Topic-word distribution

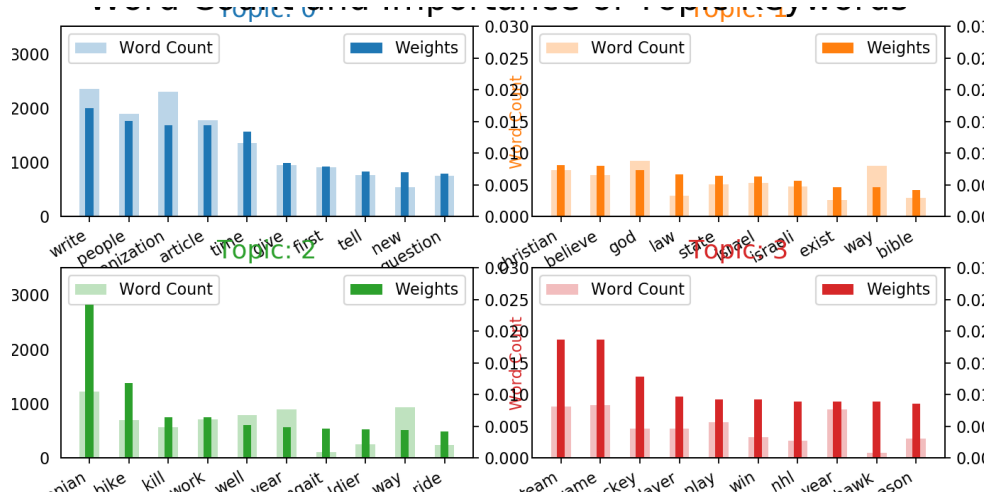


Figure 7.3 - Word count and importance of topic keywords

7.8. Community Detection

Calculate distance between users using Jensen Shannon Divergence (Table 7.2).

Table 7.2 - Distance between users

	User1	User2	User3	User4	User5
User1	0.00000	1.260869	1.163385	1.0226157	1.243853
User2	1.260869	0.00000	1.231731	1.264484	1.247072
User3	1.163385	1.231731	0.00000	1.263051	1.142907
User4	1.226157	1.264484	1.263051	0.00000	1.230949
User5	1.243853	1.247072	1.142907	1.230949	0.00000

Construct graph (Figure 7.4) based on closeness between users according to selected topic (Table 7.3).

Table 7.3 - Closeness between users for selected topic

	Topic1	Topic2	Topic3	Topic4	Topic5
User1	0.016541	0.001504	0.001504	0.978947	0.001504
User2	0.001835	0.001835	0.992661	0.001835	0.001835
User3	0.952838	0.000873	0.000873	0.044541	0.000873
User4	0.000608	0.006687	0.000608	0.012766	0.979331
User5	0.079154	0.906949	0.006647	0.000604	0.006647

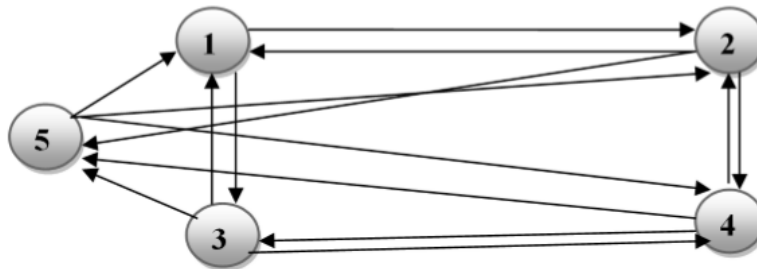


Figure 7.5 - Constructed graph

Fig. 7.5 illustrates first five most related topics in one community. Here the majority of users in this community treat topic "Music". When compare with actual group of users in this detected community, interest of actual community also closer to "Music".

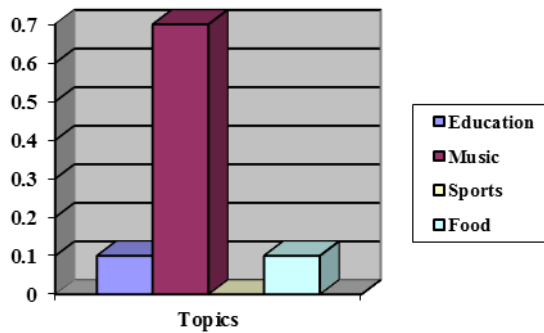


Figure 7.6 - Topics treated in community1

Table Error! No text of specified style in document.4 - Evaluate the community detection based on different number of topics for trained model

	K=10	K=20	K=40	Human
Person 1	Music	Entertainment	Travel	Entertainment
Person 2	Country	Politics	Ethics	Political Science
Person 3	Medicine	Education	Research	Higher education
Person 4	News	Sport	Gossips	Games
Person 5	Crimes	Finance	News	Business
Person 6	Education	Computer Science	Engineering	IT
Person 7	Politics	Universal Explore	Music	News
Person 8	Science	Meditation	Cultural	Religious
Person 9	Art	Movies	Gossips	Movies
Person 10	Dancing	Entertainment	Travel	Entertainment

Further Fig. 7.1 can be justified by considering data in above Table 7.4. Table 7.4 interprets the communities defined for randomly selected 10 people in the test bed for different K values given for training LDA model. Also it presents actual communities defined by human for given person based on his preferred interests. When comparing those results we can further interpret that the best LDA model creates when K=20 since it is closer to topics defined by human.

7.9. Search Optimization

1. Step 01 - Filter from User's Documents

Filter related top articles from user uploaded documents with probabilities for given search query (Table 7.5)

Table 7.5 - Filter Articles User Uploaded Wise

Article	Probability	Tag
Article_1	0.09	UserUpload
Article_2	0.085	UserUpload

2. Step 02 – Filter using Item-based collaborative filtering (Table 7.6)

Take top 3 user belonging clusters and consider users of those clusters. Consider above users' search history and filter related articles from user uploaded documents with probabilities for given search query

- User 1 - [Article_1, Article_2, Article_3]
- User 2 - [Article_1, Article_2]
- User 3 - [Article_1]

Table Error! No text of specified style in document..6 - Filter articles based on item based collaborative filtering

Article	Probability	Tag
Article_19	$f(\text{Article_19})/n(\text{articles}) = 0.01$	Collaborative
Article_29	$f(\text{Article_29})/n(\text{articles}) = 0.06$	Collaborative
Article_1	$f(\text{Article_1})/n(\text{articles}) = 0.04$	Collaborative

3. Step 04 – Prioritize articles (Table 7.7)

Table 7.7 - Prioritize search results

Priority	Article	Probability
1	Article_1	$0.09 + 0.04$
2	Article_2	0.085
5	Article_29	0.06
6	Article_19	0.01

8. CONCLUSION AND FURTHER WORK

8.1. Introduction

It is evident that this search engine framework can perform better in searching than a conventional search engine. It enables the users to identify the community preferences even there is no pre-established community topologies. Therefore, the users of internal web applications which does not have pre-defined communities, could access the relevant information which is hidden in large volumes of data without any usual hassle of searching. This makes searching a pleasant experience for users.

This framework can be integrated to individual web sites and would remarkably improve relevant search results. Further integration of personalization based on dynamically defined communities will be useful for users.

Search engines or localized software systems developed for information searching, play an important role in knowledge discovery. Proliferation of data in the web and social media has posed significant challenges in finding relevant information efficiently even using those search engines or other software systems. Moreover, those systems or engines tend to collect massive number of data, which could be useful for humans in various ways but overlook the meaning of the search phrases, hence generate irrelevant search results.

A unit level searching i.e. searching information within a website or page is also not effective as they follow exact keyword matching techniques and ignore the semantic level matching of search phrases. In order to address those deficiencies, this research proposes a hybrid approach which use the semantics of data, community preferences as well as collaborative filtering techniques for semantic information retrieval. More specifically, Topic modeling based on Latent Dirichlet Allocation together with topic-driven based community detection methods are applied for identifying personalized

search results and hence improve the relatedness of the research results. Based on the proposed hybrid approach a framework for semantic search that can easily be integrated to a software application has been implemented. The evaluation results confirm the effectiveness of search results which outperform benchmark approaches that follow traditional keyword search algorithms.

8.2. Conclusion

Proposed solution is to provide a semantic information retrieval platform for internal searching of a software application. Information retrieval would be based on semantics of data, based on topic-based community detection, based on applying personalization and collaborative filtering on top of detected communities for providing more meaningful search results to user. To provide semantically rich search results, application will look into latent Dirichlet allocation and topic-driven community detection methods integrating with collaborative filtering.

The system contains five modules called

1. UI module which provide user interface for uploading documents and entering search query. Here user can create documents and folders also. User can view, edit, delete uploaded files. User can configure the settings related to their company.
2. Discover topics Module which discover hidden topics in user uploaded documents. First the documents are preprocessed and then train LDA model and by using LDA model do topic extraction. These extracted topics are used to define communities and indexing search engine.
3. Community extraction module which is detecting communities based on topic-driven approach and community detection algorithms. Here by using LDA topic-model which returned user-topic distributions and by using Jensen-Shannon Divergence algorithm, the topic distances between two users are calculated. Then using the calculated distances, community graph has been constructed. Then to construct use

communities used an adapted approach of Girvan-Newman which is based on divisive classification.

4. Item based collaborative filtering Module which uses the detected communities and user history to find more relevant articles. Item based collaborative filtering consider about the user search history in a particular community (based on communities identified by above community detection module) and according to the frequency of any article viewed or downloaded, article would be given a preference value. These inferred preference values are stored with the search results and they are used to derive a final composite score, on which ultimate search results are based on.
5. Search optimization Module which is responsible for integrating above modules to search engine, build indexing, rank files, execute the query and find best search results. Lucene is used to internalize the topics and topic memberships while building the index and executing the queries. Here used Payloads to cleverly encode the topics in each document at index time. When user has entered the query, determine which topics are in the query, based on the terms in the query. Then create a Payload query based on these topics. Lucene will then find all documents that contain these topics. We ignore the actual relevancy returned by Lucene, and instead use the contents of the Payload to compute the relevancy ourselves, and re-rank the results.

Achievement 01 – Objective “In-depth study of technology used for semantic meta data extraction” has been achieved by in-depth study of literature review shown in Chapter 2. We have reviewed 20 articles on semantic meta data extraction on latest. Identify 3 problems and argued for 1.

Achievement 02 – Objective “Critical review of analyzing topic-based community detection” has been achieved by in depth study of literature review shown in Chapter 2. We have reviewed 15 articles on community detection methods on latest. Identify 5 problems and argued for 1.

Achievement 03 – Objective “Critical review of analyzing item based collaborative filtering” has been achieved by in depth study of literature review shown in Chapter 2. We have reviewed 10 articles on collaborative filtering methods on latest. Identify 4 problems and argued for 1.

Achievement 04 – Demonstrated in chapter 4,5,6.

Achievement 05 – Demonstrated by chapter Evaluation. Here we reported test cases, participants etc.

8.3. Further Work

The system is performing well for defining topics for given articles and cluster user communities. The system only supports for PDF, Word, XML documents is one limitation of the system. The system supports only English language. Hence topic modeling and topic-driven to detect dynamic communities approach is able to provide rich semantic search engine framework to provide best search results with personalization for a given query.

REFERENCES

- [1] Tehseen R (2018) Semantic Information Retrieval: A Survey. *J Inform Tech Softw Eng* 8: 241. DOI: 10.4172/2165-7866.1000241
- [2] M. Yadav, "A Review Paper on Information Retrieval Techniques for Point and Range Query in Database System," *International Journal of Advanced Research in Computer Science*, p. 5, 2017.
- [3] M. Bansal and J. Arora, "A Review on Ontology based Information Retrieval System," vol. 4, no. 2, p. 3, 2016.
- [4] A. P. J. M. J. P. a. J. T. F. E. A. Calvillo, "Searching research papers using clustering and text mining," in *CONIELECOMP 2013, 23rd International Conference on Electronics, Communications and Computing*, 2013.
- [5] Jayaratne, Madhura & Haththotuwa, Isuru & Dandeniya Arachchi, Charitha & Perera, Shehan & Fernando, Dilina & Weerakoon, Sajith. (2012). iSeS: Intelligent Semantic Search Framework. *Proceedings of the 6th Euro American Conference on Telematics and Information Systems*, EATIS 2012. 10.1145/2261605.2261637.
- [6] B. Dib, F. Kalloubi, E. H. Nfaoui, and A. Boulaalam, "Semantic-based Followee Recommendations on Twitter Network," *Procedia Computer Science*, vol. 127, pp. 505–510, Jan. 2018.
- [7] S. Deerwester, S. T. Dumais, G.W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the Association for Information Science and Technology*, vol. 41, no. 6, pp. 391–407, 1990.
- [8] S. Deerwester, "Indexing by latent semantic analysis," *Journal of the Association for Information Science & Technology*, vol. 41, no. 6, pp. 391–407, 2010.
- [9] T. Hofmann, "Unsupervised learning by probabilistic Latent Semantic Analysis," *Machine Learning*, vol. 42, no. 1-2, pp. 177–196, 2001.
- [10] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.
- [11] W. Ding and Zhaoyun, "Mining user interest in microblogs with a user-topic model," *China Communications*, vol. 11, no. 8, pp. 131–144, 2014.
- [12] T. Hofmann, "Probabilistic latent semantic indexing," in *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '99)*, pp. 50–57, Berkeley, Calif, USA, 1999.
- [13] S. P. Y. Xing Pingping, "Ontology-based Data Mining," 2001.
- [14] G. K. J. K. a. J. R. B. Sarwar, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the tenth international conference on World Wide Web - WWW, Hong Kong*, 2001.
- [15] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.
- [16] D. M. Blei, "Probabilistic topic models," *Commun. ACM*, vol. 55, no. 4, pp. 77–84, 2012.

- [17]D. L. a. A. Maclachlan, "Slope One Predictors for Online Rating-Based Collaborative Filtering," in In Siam Data Mining, 2005
- [18]M.Sachan,D.Contractor,T.A.Faruquie,andL.V.Subramaniam,“Using content and interactions for discovering communities in social networks,” in *Proc. 21st Int. Conf. World Wide Web*, 2012, pp. 331–340.
- [19]N.Pathak,C.DeLong,A.Banerjee,andK.Erickson,“Socialtopicmodels for community extraction,” in Proc. 2nd SNA-KDD Workshop, 2008, pp. 1–10
- [20]K.LimandA.Datta,“A topological approach for detecting twitter communities with common interests,” in Ubiquitous Social Media Analysis (Lecture Notes in Computer Science), vol. 8329. Berlin, Germany: Springer, 2013, pp. 23–43
- [21]L. Hannachi, O. Asfari, N. Benblidia, F. Bentayeb, N. Kabachi, and O. Boussaid, “Community extraction based on topic-driven-model for clustering user tweets,” in Advanced Data Mining and Applications (Lecture Notes in Computer Science), vol. 7713. Berlin, Germany: Springer, 2012, pp. 39–51.
- [22]S. Jaffali, S. Jamoussi, and A. B. Hamadou, “Grouping like-minded users based on text and sentiment analysis,” in Computational Collective Intelligence. Technologies and Applications (Lecture Notes in Computer Science), vol. 8733. Berlin, Germany: Springer, 2014, pp. 83–93
- [23]Weng, J., Lim, E.P., Jiang, J., He, Q.: Twitterrank: finding topic-sensitive influential twitterers. In: WSDM, pp. 261–270 (2010)
- [24]R. I. M. Dunbar, “Do online social media cut through the constraints that limit the size of offline social networks?” *Roy. Soc. OpenSci.*, vol.3,no.1, p. 150292, Feb. 2016.
- [25]“Apache Lucene - Welcome to Apache Lucene.” [Online]. Available: <http://lucene.apache.org/>. [Accessed: 22-Feb-2019].
- [26]“Index of /simplewiki/.” [Online]. Available: <https://dumps.wikimedia.org/simplewiki/>. [Accessed: 21-Feb-2019].
- [27]T.Berner-Lee and M. Fishetti, Weaving the web “chapter Machines and the web,”Chapter Machines and the web, pp. 177-198, 1999.
- [28]D.Fensal, W. Wahlster, H. Lieberman, "Spanning the semantic web: Bringing the worldwide web to its full potential," MIT Press 2003.
- [29]G. Bholotia et al.: “Keyword searching and browsing in database using BANKS,” 18th Intl. conf. on Data Engineering (ICDE 2002), San Jose, USA, 2002.
- [30]D. Tümer, M. A. Shah, and Y. Bitirim, An Empirical Evaluation on Semantic Search Performance of Keyword-Based and Semantic Search Engines: Google, Yahoo, Msn and Hakia, 2009 *4th International Conference on Internet Monitoring and Protection (ICIMP '09)* 2009.
- [31]"Top 5 Semantic Search Engines".<http://www.pandia.com/>.
- [32]H. Dietze and M. Schroeder, GoWeb: a semantic search engine for the life science web . *BMC bioinformatics* ,Vol. 10, No. Suppl 10, pp. S7, 2009.

- [33] Sanjib kumar, Sanjay kumar malik "TOWARDS SEMANTIC WEB BASED SEARCH ENGINES" *National Conference on "Advances in Computer Networks & Information Technology (NCACNIT-09)* March 24-25,
- [34] F. F. Ramos, H. Unger, V. Larios (Eds.): LNCS 3061, pp. 145–157, Springer-Verlag Berlin Heidelberg 2004.
- [35] Cohen, S. Mamou, J. Kanza, Y. Sagiv, Y "XSEarch: A Semantic Search Engine for XML" proceedings of the *international conference on very large databases*, pages 45-56, 2003.
- [36] D. Bhagwat and N. Polyzotis, "Searching a file system using inferred semantic links," in *Proceedings of HYPERTEXT '05 Salzburg*, 2005, pp. 85-87.
- [37] H. L. Wang, S. H. Wu, I. C. Wang, C. L. Sung, W. L. Hsu, and W. K. Shih, "Semantic search on Internet tabular information extraction for answering queries," in *Proceedings of CIKM '00 McLean*, 2000, pp.243-249.
- [38] E. Kandogan, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu, "Avatar semantic search: a database approach to information retrieval," in *Proceedings of SIGMOD '06 Chicago*, 2006, pp. 790-792.
- [39] A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz, "An infrastructure for searching, reusing and evolving distributed ontologies," in *Proceedings of WWW '03 Budapest*, 2003, pp. 439-448.
- [40] www.georges.gardarin.free.fr/Articles/Sewise_NLDB2003.pdf.

Appendix A – Python Code for Data Preprocessing

```
import xml.etree.ElementTree as ET
import codecs
import re

def is_ascii(s):
    return all(ord(c) < 128 for c in s)

tree = ET.parse('simplewiki-20170201-pages-articles-multistream.xml')
root = tree.getroot()

dir_path = 'articles-corpus//'

for i,page in
enumerate(root.findall('{http://www.mediawiki.org/xml/export-
0.10/}page')):
    for p in page:
        if p.tag == "{http://www.mediawiki.org/xml/export-0.10/}revision":
            for x in p:
                if x.tag == "{http://www.mediawiki.org/xml/export-
0.10/}text":
                    article_txt = x.text
                    if not article_txt == None:
                        article_txt = article_txt[ :
article_txt.find("==")]
                        article_txt = re.sub(r"{{.*}}", "", article_txt)
                        article_txt =
re.sub(r"\[\[File:.*\]\]", "", article_txt)
                        article_txt =
re.sub(r"\[\[Image:.*\]\]", "", article_txt)
                        article_txt = re.sub(r"\n: \'\'.*", "", article_txt)
                        article_txt = re.sub(r"\n!.*", "", article_txt)
                        article_txt = re.sub(r"^\:\'\'.*", "", article_txt)
                        article_txt = re.sub(r"&nbsp;", "", article_txt)
                        article_txt = re.sub(r"http\S+", "", article_txt)
                        article_txt = re.sub(r"\d+", "", article_txt)
                        article_txt = re.sub(r"\(.*\)", "", article_txt)
                        article_txt =
re.sub(r"Category:.*", "", article_txt)
                        article_txt = re.sub(r"\| .*", "", article_txt)
                        article_txt = re.sub(r"\n\|.*", "", article_txt)
```

```

        article_txt = re.sub(r"\n \|.*", "", article_txt)
        article_txt = re.sub(r".* \|\n", "", article_txt)
        article_txt = re.sub(r".*\|\n", "", article_txt)
        article_txt =
re.sub(r"{{Infobox.*", "", article_txt)
        article_txt =
re.sub(r"{{infobox.*", "", article_txt)
        article_txt =
re.sub(r"{{taxobox.*", "", article_txt)
        article_txt =
re.sub(r"{{Taxobox.*", "", article_txt)
        article_txt = re.sub(r"{{
Infobox.*", "", article_txt)
        article_txt = re.sub(r"{{
infobox.*", "", article_txt)
        article_txt = re.sub(r"{{
taxobox.*", "", article_txt)
        article_txt = re.sub(r"{{
Taxobox.*", "", article_txt)
        article_txt = re.sub(r"\* .*", "", article_txt)
        article_txt = re.sub(r"<.*>", "", article_txt)
        article_txt = re.sub(r"\n", "", article_txt)
        article_txt =
re.sub(r"!|\\"|\#|\$|\%|\&|\'|\\(|\)|\*|\+|\,|\-
|\.|\||\:\|;\|<|\=|\>|\?|\@|\[|\]|\\|\]|\\^|\_|\`|\{|\}|\||\~", "
", article_txt)
        article_txt = re.sub(r" +", " ", article_txt)
        article_txt = article_txt.replace(u'\xa0', u' ')

        if not article_txt == None and not article_txt ==
"" and len(article_txt) > 150 and is_ascii(article_txt):
            outfile = dir_path + str(i+1) + "_article.txt"
            f = codecs.open(outfile, "w", "utf-8")
            f.write(article_txt)
            f.close()
            print article_txt
            print
'\n=====\\n'

```

```
import os
```



```

import random
import codecs
import cPickle
from gensim.models.ldamodel import LdaModel as Lda
from gensim import corpora
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer

# Function to remove stop words from sentences & lemmatize verbs.
def clean(doc):
    stop_free = " ".join([i for i in doc.lower().split() if i not in
stop])
    normalized = " ".join(lemma.lemmatize(word,'v') for word in
stop_free.split())
    x = normalized.split()
    y = [s for s in x if len(s) > 2]
    return y

corpus_path = "articles-corpus/"
article_paths = [os.path.join(corpus_path,p) for p in
os.listdir(corpus_path)]

# Read contents of all the articles in a list "doc_complete"
doc_complete = []
for path in article_paths:
    fp = codecs.open(path,'r','utf-8')
    doc_content = fp.read()
    doc_complete.append(doc_content)

# Randomly sample 70000 articles from the corpus created from the 1st
blog-post (wiki_parser.py)
docs_all = random.sample(doc_complete, 60000)
docs = open("docs_wiki.pkl",'wb')
cPickle.dump(docs_all,docs)

# Use 60000 articles for training.
docs_train = docs_all[:50000]

# Cleaning all the 60,000 simplewiki articles
stop = set(stopwords.words('english'))
lemma = WordNetLemmatizer()

```

```

doc_clean = [clean(doc) for doc in docs_train]

# Creating the term dictionary of our corpus, where every unique term is
# assigned an index.
dictionary = corpora.Dictionary(doc_clean)

# Filter the terms which have occurred in less than 3 articles and more
# than 40% of the articles
dictionary.filter_extremes(no_below=4, no_above=0.4)

# List of some words which has to be removed from dictionary as they are
# content neutral words
stoplist = set('also use make people know many call include part find
become like mean often different \
            usually take wikt come give well get since type list say
change see refer actually iii \
            aisne kinds pas ask would way something need things want
every str'.split())
stop_ids = [dictionary.token2id[word] for word in stoplist if
word in dictionary.token2id]
dictionary.filter_tokens(stop_ids)

#words,ids = dictionary.filter_n_most_frequent(50)
#print words,"\n\n",ids

# Converting list of documents (corpus) into Document Term Matrix using
# dictionary prepared above.
doc_term_matrix = [dictionary.doc2bow(doc) for doc in doc_clean]

```

Appendix B – Calculate Distance between users Using Jensen Shannon Divergence

```
import random
import pandas as pd
import json
import os
import ast
import sys
import argparse
import argparse
import scipy
import tqdm
from shutil import copyfile
from scipy.linalg import norm
from scipy.stats import entropy
import multiprocessing
from functools import partial
import numpy as np
import matplotlib.pyplot as plt
from collections import defaultdict
from gensim import corpora, models, matutils

def calculate_distances(community):
    """
    for each user find the distance from every other user using their
    probability distribution vectors

    Dictionary <k, v>(user_id, distribution_vector)

    """
    distance_dir = os.path.join(community, 'calculated_distances/')
    if(os.path.exists(os.path.join(distance_dir,
'median_community_distances'))): return
    comm_doc_vecs =
open_community_document_vectors_file(os.path.join(community,
'community_doc_vecs.json'))
    if(len(comm_doc_vecs) <= 1): return
    if not os.path.exists(os.path.dirname(distance_dir)):
        os.makedirs(os.path.dirname(distance_dir), 0o755)
```

```

jen_shan_file = os.path.join(distance_dir, 'jensen_shannon')
if os.path.exists(jen_shan_file): os.remove(jen_shan_file)
with open(jen_shan_file, 'w') as out:
    for key in sorted(comm_doc_vecs):
        user = key
        # only necessary to compare each user with any other user once
        vec_1 = comm_doc_vecs.pop(key)

        for key_2 in sorted(comm_doc_vecs):
            vec_2 = comm_doc_vecs[key_2]
            out.write('{}\t{}\t{}\n'.format(user, key_2,
jensen_shannon_divergence(vec_1, vec_2)))

def jensen_shannon_divergence(P, Q):
    _P = np.array(P) / norm(np.array(P), ord=1)
    _Q = np.array(Q) / norm(np.array(Q), ord=1)
    _M = 0.5 * (_P + _Q)
    return 0.5 * (entropy(_P, _M) + entropy(_Q, _M))

def individual_user_distance_graphs(internal, community):
    '''
    creates graph displaying each user in the community comparing the
    jensen shannon divergences between
    their probability distribution vectors against other users

    x-axis: users in community, y-axis: distance from observed user
    '''
    distance_dir = os.path.join(community, 'calculated_distances/')
    if internal:
        jsd_dists = os.path.join(distance_dir, 'jensen_shannon')
        out_path = os.path.join(os.path.join(community,
'internal_user_graphs/jensen_shannon/'))
        out_file = os.path.join(distance_dir, 'community_distances')
    else:
        jsd_dists = os.path.join(distance_dir, 'jensen_shannon_ext')
        out_path = os.path.join(os.path.join(community,
'external_user_graphs/jensen_shannon/'))
        out_file = os.path.join(distance_dir, 'ext_community_distances')
    comm_doc_vecs =
open_community_document_vectors_file(os.path.join(community,
'community_doc_vecs.json'))
    if(len(comm_doc_vecs) <= 1): return

```

```

    if not os.path.exists(os.path.dirname(out_path)):
        os.makedirs(os.path.dirname(out_path), 0o755)
    x_axis = np.arange(1, len(comm_doc_vecs))
    df = pd.read_csv(jsd_dists, sep='\t', header=None, names=['user_1',
'user_2', 'distance'])
    for user in comm_doc_vecs:
        if not os.path.exists(os.path.join(out_path, user + '.png')):
            new_df = df[(df.user_1 == int(user)) | (df.user_2 ==
int(user))]
            new_df.to_csv(out_path + str(user), sep='\t', header=None,
index=None)
            y_axis = new_df['distance'].tolist()
            draw_scatter_graph(user, 'Community Members', 'Jensen Shannon
Divergence', x_axis, y_axis, 0, len(x_axis) + 1, 0, (np.log(2) + 0.1),
os.path.join(out_path, user))

def draw_scatter_graph(title, x_label, y_label, x_axis, y_axis, min_x,
max_x, min_y, max_y, output_path):
    fig = plt.figure()
    fig.suptitle(title, fontsize=14, fontweight='bold')
    ax = fig.add_subplot(111)
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    ax.plot(x_axis, y_axis, 'o')
    ax.axis([min_x, max_x, min_y, max_y])
    plt.margins(0.2)
    plt.tick_params(labelsize=10)
    fig.subplots_adjust(bottom=0.2)
    plt.savefig(output_path)
    plt.close(fig)

```

Appendix C – Construct the Communities using Girvan Newman Algorithm

```
#!/usr/bin/env python
import networkx as nx
import math
import csv
import random as rand
import sys

_DEBUG_ = False

#this method just reads the graph structure from the file
def buildG(G, file_, delimiter_):
    #construct the weighted version of the contact graph from cgraph.dat
    file
    #reader = csv.reader(open("/home/kazem/Data/UCI/karate.txt"),
    delimiter=" ")
    reader = csv.reader(open(file_), delimiter=delimiter_)
    for line in reader:
        if len(line) > 2:
            if float(line[2]) != 0.0:
                #line format: u,v,w
                G.add_edge(int(line[0]),int(line[1]),weight=float(line[2]))
            else:
                #line format: u,v
                G.add_edge(int(line[0]),int(line[1]),weight=1.0)

#keep removing edges from Graph until one of the connected components of
Graph splits into two
#compute the edge betweenness
def CmtyGirvanNewmanStep(G):
    if _DEBUG_:
        print "Calling CmtyGirvanNewmanStep"
    init_ncomp = nx.number_connected_components(G)    #no of components
    ncomp = init_ncomp
    while ncomp <= init_ncomp:
        bw = nx.edge_betweenness_centrality(G, weight='weight')    #edge
betweenness for G
        #find the edge with max centrality
```

```

        max_ = max(bw.values())
        #find the edge with the highest centrality and remove all of them
if there is more than one!
        for k, v in bw.iteritems():
            if float(v) == max_:
                G.remove_edge(k[0],k[1])    #remove the central edge
            ncomp = nx.number_connected_components(G)    #recalculate the no
of components

#compute the modularity of current split
def _GirvanNewmanGetModularity(G, deg_, m_):
    New_A = nx.adj_matrix(G)
    New_deg = {}
    New_deg = UpdateDeg(New_A, G.nodes())
    #Let's compute the Q
    comps = nx.connected_components(G)    #list of components
    print 'No of communities in decomposed G: %d' %
nx.number_connected_components(G)
    Mod = 0    #Modularity of a given partitioning
    for c in comps:
        EWC = 0    #no of edges within a community
        RE = 0    #no of random edges
        for u in c:
            EWC += New_deg[u]
            RE += deg_[u]    #count the probability of a random edge
        Mod += ( float(EWC) - float(RE*RE)/float(2*m_) )
    Mod = Mod/float(2*m_)
    if _DEBUG_:
        print "Modularity: %f" % Mod
    return Mod

def UpdateDeg(A, nodes):
    deg_dict = {}
    n = len(nodes)    #len(A) ---> some ppl get issues when trying len() on
sparse matrixes!
    B = A.sum(axis = 1)
    for i in range(n):
        deg_dict[nodes[i]] = B[i, 0]
    return deg_dict

#run GirvanNewman algorithm and find the best community split by
maximizing modularity measure

```

```

def runGirvanNewman(G, Orig_deg, m_):
    #let's find the best split of the graph
    BestQ = 0.0
    Q = 0.0
    while True:
        CmtyGirvanNewmanStep(G)
        Q = _GirvanNewmanGetModularity(G, Orig_deg, m_);
        print "Modularity of decomposed G: %f" % Q
        if Q > BestQ:
            BestQ = Q
            Bestcomps = nx.connected_components(G)    #Best Split
            print "Components:", Bestcomps
            if G.number_of_edges() == 0:
                break
    if BestQ > 0.0:
        print "Max modularity (Q): %f" % BestQ
        print "Graph communities:", Bestcomps
    else:
        print "Max modularity (Q): %f" % BestQ

def main(argv):
    if len(argv) < 2:
        sys.stderr.write("Usage: %s <input graph>\n" % (argv[0],))
        return 1
    graph_fn = argv[1]
    G = nx.Graph() #let's create the graph first
    buildG(G, graph_fn, ',')

    if _DEBUG_:
        print 'G nodes:', G.nodes()
        print 'G no of nodes:', G.number_of_nodes()

    n = G.number_of_nodes()    #|V|
    A = nx.adj_matrix(G)    #adjacencnt matrix

    m_ = 0.0    #the weighted version for number of edges
    for i in range(0,n):
        for j in range(0,n):
            m_ += A[i,j]
    m_ = m_/2.0
    if _DEBUG_:
        print "m: %f" % m_

```



```
#calculate the weighted degree for each node
Orig_deg = {}
Orig_deg = UpdateDeg(A, G.nodes())

#run Newman alg
runGirvanNewman(G, Orig_deg, m_)

if __name__ == "__main__":
    sys.exit(main(sys.argv))
```

Appendix D – Item Based Collaborative Filtering which returns Recommended Articles for Particular User

```
import numpy as np
import scipy.stats
import scipy.spatial
from sklearn.cross_validation import KFold
import random
from sklearn.metrics import mean_squared_error
from math import sqrt
import math
import warnings
import sys
#from sklearn.utils.extmath import np.dot

warnings.simplefilter("error")

users = 6040
items = 3952

def readingFile(filename):
    f = open(filename,"r")
    data = []
    for row in f:
        r = row.split(',')
        e = [int(r[0]), int(r[1]), int(r[2])]
        data.append(e)
    return data

def similarity_item(data):
    print "Hello"
    #f_i_d = open("sim_item_based.txt","w")
    item_similarity_cosine = np.zeros((items,items))
    item_similarity_jaccard = np.zeros((items,items))
    item_similarity_pearson = np.zeros((items,items))
    for item1 in range(items):
        print item1
        for item2 in range(items):
            if np.count_nonzero(data[:,item1]) and
np.count_nonzero(data[:,item2]):
```

```

        item_similarity_cosine[item1][item2] = 1-
scipy.spatial.distance.cosine(data[:,item1],data[:,item2])
        item_similarity_jaccard[item1][item2] = 1-
scipy.spatial.distance.jaccard(data[:,item1],data[:,item2])
    try:
        if not
math.isnan(scipy.stats.pearsonr(data[:,item1],data[:,item2])[0]):
            item_similarity_pearson[item1][item2] =
scipy.stats.pearsonr(data[:,item1],data[:,item2])[0]
        else:
            item_similarity_pearson[item1][item2] = 0
    except:
        item_similarity_pearson[item1][item2] = 0

        #f_i_d.write(str(item1) + "," + str(item2) + "," +
str(item_similarity_cosine[item1][item2]) + "," +
str(item_similarity_jaccard[item1][item2]) + "," +
str(item_similarity_pearson[item1][item2]) + "\n")
    #f_i_d.close()
    return item_similarity_cosine, item_similarity_jaccard,
item_similarity_pearson

def crossValidation(data):
    k_fold = KFold(n=len(data), n_folds=10)

    Mat = np.zeros((users,items))
    for e in data:
        Mat[e[0]-1][e[1]-1] = e[2]

    sim_item_cosine, sim_item_jaccard, sim_item_pearson =
similarity_item(Mat)
    #sim_item_cosine, sim_item_jaccard, sim_item_pearson =
np.random.rand(items,items), np.random.rand(items,items),
np.random.rand(items,items)

    '''sim_item_cosine = np.zeros((items,items))
sim_item_jaccard = np.zeros((items,items))
sim_item_pearson = np.zeros((items,items))
f_sim_i = open("sim_item_based.txt", "r")
for row in f_sim_i:
    r = row.strip().split(',')
    sim_item_cosine[int(r[0])][int(r[1])] = float(r[2])
'''

```

```

        sim_item_jaccard[int(r[0])][int(r[1])] = float(r[3])
        sim_item_pearson[int(r[0])][int(r[1])] = float(r[4])
    f_sim_i.close()'''

rmse_cosine = []
rmse_jaccard = []
rmse_pearson = []

for train_indices, test_indices in k_fold:
    train = [data[i] for i in train_indices]
    test = [data[i] for i in test_indices]

    M = np.zeros((users,items))

    for e in train:
        M[e[0]-1][e[1]-1] = e[2]

    true_rate = []
    pred_rate_cosine = []
    pred_rate_jaccard = []
    pred_rate_pearson = []

    for e in test:
        user = e[0]
        item = e[1]
        true_rate.append(e[2])

        pred_cosine = 3.0
        pred_jaccard = 3.0
        pred_pearson = 3.0

    #item-based
    if np.count_nonzero(M[:,item-1]):
        sim_cosine = sim_item_cosine[item-1]
        sim_jaccard = sim_item_jaccard[item-1]
        sim_pearson = sim_item_pearson[item-1]
        ind = (M[user-1] > 0)
        #ind[item-1] = False
        normal_cosine = np.sum(np.absolute(sim_cosine[ind]))
        normal_jaccard = np.sum(np.absolute(sim_jaccard[ind]))
        normal_pearson = np.sum(np.absolute(sim_pearson[ind]))
        if normal_cosine > 0:

```

```

        pred_cosine = np.dot(sim_cosine,M[user-
1])/normal_cosine

        if normal_jaccard > 0:
            pred_jaccard = np.dot(sim_jaccard,M[user-
1])/normal_jaccard

        if normal_pearson > 0:
            pred_pearson = np.dot(sim_pearson,M[user-
1])/normal_pearson

    if pred_cosine < 0:
        pred_cosine = 0

    if pred_cosine > 5:
        pred_cosine = 5

    if pred_jaccard < 0:
        pred_jaccard = 0

    if pred_jaccard > 5:
        pred_jaccard = 5

    if pred_pearson < 0:
        pred_pearson = 0

    if pred_pearson > 5:
        pred_pearson = 5

    print str(user) + "\t" + str(item) + "\t" + str(e[2]) + "\t" +
str(pred_cosine) + "\t" + str(pred_jaccard) + "\t" + str(pred_pearson)
    pred_rate_cosine.append(pred_cosine)
    pred_rate_jaccard.append(pred_jaccard)
    pred_rate_pearson.append(pred_pearson)

    rmse_cosine.append(sqrt(mean_squared_error(true_rate,
pred_rate_cosine)))
    rmse_jaccard.append(sqrt(mean_squared_error(true_rate,
pred_rate_jaccard)))
    rmse_pearson.append(sqrt(mean_squared_error(true_rate,
pred_rate_pearson)))

```

```

        print str(sqrt(mean_squared_error(true_rate, pred_rate_cosine))) +
"\t" + str(sqrt(mean_squared_error(true_rate, pred_rate_jaccard))) + "\t"
+ str(sqrt(mean_squared_error(true_rate, pred_rate_pearson)))
        #raw_input()

    #print sum(rms) / float(len(rms))
    rmse_cosine = sum(rmse_cosine) / float(len(rmse_cosine))
    rmse_pearson = sum(rmse_pearson) / float(len(rmse_pearson))
    rmse_jaccard = sum(rmse_jaccard) / float(len(rmse_jaccard))

    print str(rmse_cosine) + "\t" + str(rmse_jaccard) + "\t" +
str(rmse_pearson)

    f_rmse = open("rmse_item.txt", "w")
    f_rmse.write(str(rmse_cosine) + "\t" + str(rmse_jaccard) + "\t" +
str(rmse_pearson) + "\n")

    rmse = [rmse_cosine, rmse_jaccard, rmse_pearson]
    req_sim = rmse.index(min(rmse))

    print req_sim
    f_rmse.write(str(req_sim))
    f_rmse.close()

    if req_sim == 0:
        sim_mat_item = sim_item_cosine

    if req_sim == 1:
        sim_mat_item = sim_item_jaccard

    if req_sim == 2:
        sim_mat_item = sim_item_pearson

    #predictRating(Mat, sim_mat_item)
    return Mat, sim_mat_item

def predictRating(recommend_data):

    M, sim_item = crossValidation(recommend_data)

    #f = open("toBeRated.csv", "r")

```

```

f = open(sys.argv[2],"r")
toBeRated = {"user":[], "item":[]}
for row in f:
    r = row.split(',')
    toBeRated["item"].append(int(r[1]))
    toBeRated["user"].append(int(r[0]))

f.close()

pred_rate = []

#fw = open('result2.csv','w')
fw_w = open('result2.csv','w')

l = len(toBeRated["user"])
for e in range(l):
    user = toBeRated["user"][e]
    item = toBeRated["item"][e]

    pred = 3.0

    #item-based
    if np.count_nonzero(M[:,item-1]):
        sim = sim_item[item-1]
        ind = (M[user-1] > 0)
        #ind[item-1] = False
        normal = np.sum(np.absolute(sim[ind]))
        if normal > 0:
            pred = np.dot(sim,M[user-1])/normal

    if pred < 0:
        pred = 0

    if pred > 5:
        pred = 5

    pred_rate.append(pred)
    print str(user) + "," + str(item) + "," + str(pred)
    #fw.write(str(user) + "," + str(item) + "," + str(pred) + "\n")
    fw_w.write(str(pred) + "\n")

#fw.close()

```

```
fw_w.close()

#recommend_data = readingFile("ratings.csv")
recommend_data = readingFile(sys.argv[1])
#crossValidation(recommend_data)
predictRating(recommend_data)
```


Appendix E – Integrate with Lucene

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.TreeSet;

import org.apache.log4j.Logger;
import org.apache.lucene.document.Document;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopDocs;

class LDAHelper : Serializable {

    private static final long serialVersionUID = -5883161587870956703L;
    private static final Logger logger = Logger.getRootLogger();

    // 'scens' (short for scenarios) holds all the LDK objects: one for
each K
    // (The data structure is a simple class defined below)
    public ArrayList<LDK> scens;

    // Used when LDA needs to be run on the given inDirName
    LDAHelper(String inDirName) throws IOException{
    }

    public void runLDA(int K, String inDirName){
        LDK ldak = new LDK();
        ldak.K = K;
    }
}
```

```

        //TODO: run MALLET
        //TODO: transform MALLET datastructures into LDAK datastructures
    }

    // Used when LDA has already been run by the user, and we just need to
    // swallow up the data in the files
    LDAHelper(){
        scens = new ArrayList<LDAK>();
    }

    // Add a scenario from disk
    public void addScenario(int K, String inDirName) throws IOException{

        logger.info("Adding LDA scenario: K="+K+", dir="+inDirName);

        // First, check that the four files are present:
        if (! (new File((inDirName + "/vocab.dat")).exists())){
            System.err.println("Error: " + inDirName + "/vocab.dat does
not exist.");
            return;
        }
        if (! (new File((inDirName + "/files.dat")).exists())){
            System.err.println("Error: " + inDirName + "/files.dat does
not exist.");
            return;
        }
        if (! (new File((inDirName + "/theta.dat")).exists())){
            System.err.println("Error: " + inDirName + "/theta.dat does
not exist.");
            return;
        }
        if (! (new File((inDirName + "/words.dat")).exists())){
            System.err.println("Error: " + inDirName + "/words.dat does
not exist.");
            return;
        }

        LDAK ldak = new LDAK();
        ldak.K = K;

        // Read the term map

```

```

        BufferedReader br = new BufferedReader(new FileReader(inDirName +
"/vocab.dat"));
        int counter=0;
        String line;
        while ((line = br.readLine()) != null) {
            ldak.termMap.put(line, counter);
            ++counter;
        }

        // Read the file map
        br = new BufferedReader(new FileReader(inDirName + "/files.dat"));
        counter=0;
        while ((line = br.readLine()) != null) {
            String lineParts[] = line.split("\\s+");
            ldak.fileMap.put(lineParts[1], counter);
            ++counter;
        }

        // Set the constants, that will be used later.
        ldak.D = ldak.fileMap.size();
        ldak.W = ldak.termMap.size();

        // Read the theta and phi matrices
        ldak.theta = readFileIntoMatrix(inDirName + "/theta.dat", ldak.D,
ldak.K);
        ldak.phi = readFileIntoMatrix(inDirName + "/words.dat", ldak.K,
ldak.W);

        scens.add(ldak);
    }

    /**
     *
     * @param fileName
     * @param numRows
     * @param numCols
     * @return
     */
    private float[][] readFileIntoMatrix(String fileName, int numRows, int
numCols) {
        BufferedReader br = null;

```

```

float[][] matrix = new float[numRows][numCols];

try {
    br = new BufferedReader(new FileReader(fileName));
    String line = null;
    int x=0;
    int y=0;
    while ((line = br.readLine()) != null) {
        String[] values = line.split("\\s+");
        y = 0;

        // Special case: MALLET returns nans, for example if there
were more topics
        // than documents. In this case, just make a row of 0s
        if (values[0].equals("nan")){
            for (String str : values) {
                matrix[x][y++]=0;
            }
        } else {
            // Normal case: everything went as expected (real
valued numbers
            // in file)
            for (String str : values) {
                float str_double = Float.parseFloat(str);
                matrix[x][y++]=str_double;
            }
        }
        ++x;
    }
} catch (Exception e){
    e.printStackTrace();
    return null;
}

return matrix;
}

public String encodeTopics(int docId, int K) {
    String out = "";
    int idx = which(K);

```

```

        for (int i=0;i<scens.get(idx).K;++i){
            out += ("," + scens.get(idx).theta[docId][i]);
        }
        return out;
    }

    // Given an K value, this function returns the index of this K in the
    // scens ArrayList.
    public int which(int k) {

        // Special case: if k==0, then the command line option was
        // omitted and we should return the
        // index of the first k.
        if (k==0){
            return 1;
        }

        for (int i = 0; i < scens.size(); ++i){
            if (k == scens.get(i).K){
                return i;
            }
        }

        // Default: just return the index of the first K.
        return 1;
    }

    public float[] decodeTopics(String encodedTopicString, int K) {
        String[] parts = encodedTopicString.split(",");
        int idx = which(K);
        float result[] = new float[scens.get(idx).K];
        for (int i=1;i<parts.length;++i){
            result[i-1] = Float.parseFloat(parts[i]);
        }
        return result;
    }

    public String encodeTopicsPayload(int docId, int K) {
        String out = "";
        int idx = which(K);

```

```

        for (int i=0;i<scens.get(idx).K;++i){
            if (scens.get(idx).theta[docId][i] > 0.05){
                out += (" p" + i + "$" + scens.get(idx).theta[docId][i]);
            }
        }
        return out;
    }

    /**
     *
     * @param searcher
     * @param hits
     * @param queryScore
     * @param K
     * @return
     * @throws IOException
     * @throws Exception
     */
    public HashMap<String, Float> reRank(IndexSearcher searcher, TopDocs
hits, float queryScore[], int K) throws IOException, Exception{
        ScoreDoc[] scoreDocs = hits.scoreDocs;

        HashMap<String, Float> result = new HashMap<String, Float>();

        for (int n = 0; n < scoreDocs.length; ++n) {
            ScoreDoc sd = scoreDocs[n];
            int docId = sd.doc;
            Document d = searcher.doc(docId);
            String fileName = d.get("file");

            String encodedTopicString = (d.get("topics" + K));
            float sim = computeSimilarity(encodedTopicString, queryScore,
K);

            //System.out.printf("%3d %4.5f %d %s\n", n, sim, docId,
fileName);
            result.put(fileName, sim);
        }

        return sortHashMap(result);
    }
}

```

```

/**
 *
 * @param input
 * @return
 */
private HashMap<String, Float> sortHashMap(HashMap<String, Float>
input){
    Map<String, Float> tempMap = new HashMap<String, Float>();
    for (String wsState : input.keySet()){
        tempMap.put(wsState,input.get(wsState));
    }

    List<String> mapKeys = new ArrayList<String>(tempMap.keySet());
    List<Float> mapValues = new ArrayList<Float>(tempMap.values());
    HashMap<String, Float> sortedMap = new LinkedHashMap<String,
Float>();
    TreeSet<Float> sortedSet = new TreeSet<Float>(mapValues);
    Object[] sortedArray = sortedSet.descendingSet().toArray();
    int size = sortedArray.length;
    for (int i=0; i<size; i++){
        sortedMap.put(mapKeys.get(mapValues.indexOf(sortedArray[i])),
            (Float)sortedArray[i]);
    }
    return sortedMap;
}

/**
 *
 * @param docId
 * @param queryScore
 * @return The conditional probability between the document and the
query.
 */
public float computeSimilarity(String encodedTopicString, float
queryScore[], int K){
    float result = 0;
    float[] topicVector = decodeTopics(encodedTopicString, K);
    for (int i=0; i<topicVector.length;++i){
        result += topicVector[i]*queryScore[i];
    }
}

```

```
    return result;
}

// Data containers
public class LDAK implements Serializable {
    private static final long serialVersionUID = 2161745883533541761L;
    public float phi[][]; // holds all the topics
    public float theta[][]; // holds all the topic vectors
    public int K = 0; // The number of LDA topics
    public int W = 0; // The number of terms
    public int D = 0; // number of documents
    public HashMap<String, Integer> termMap = new HashMap<String,
Integer>(); // Contains the ids of each term (for the topics)
    public HashMap<String, Integer> fileMap = new HashMap<String,
Integer>(); // Contains the file map
}
}
```