Data Mining Algorithms in SF1-R

Jinglei Zhao, Jia Guo

iZENEsoft, Inc.

Technical Report in iZENEsoft.

Abstract

SF1-revolution includes many data mining components to increase the effectiveness of searching to better sasify users' information need. This technical report gives a detailed introduction to the core algorithm designs in SF1 revolution.

Contents

Chapter 1

Introduction

In enterprises and organizations, the creation of data is increasing at gigantic rate every day. The enormous growth comes from a variety of different sources such as file systems, intranets, document management systems, email and databases. An effective access to the needed information could increase the business opportunities and raise the productivity, which has become a key to the success of business.

By using search engines, a user first selects the natural language terms to express their information need and search against the indexed data to find the relevant documents. A good search engine should provide the most relevant information that best satisfy the user's information need.

Search technologies have been advanced for several decades to bring the most relevant information to the user. There are three stages that we could identify in the evolution of search engines.

- The first generation.

  Uses mostly on-page data (text and format) to determine a document's relevance. Typically, keyword matching is done between a user query and the indexed documents to bring the matched documents to the user.

- The second generation.

Uses additional off-page data such as link analysis, document click data to help bring the relevant documents to a query. Link analysis has bring much advancement in Web search. By now, all major Web search engines uses link data.

- The third generation.

  Attempts to blend data from multiple sources in order to try to answer "the need behind the query". For example, on a query like *San Francisco*, the engine might present representative aspects for San Francisco such as hotel, tourism guide etc. Thus, third generation engines go beyond the keyword matching by providing more informative results through massive semantic analysis, data mining etc.

Currently, most enterprise search engines are still built upon the first generation and second generation technology based mainly on keyword matching. However, in many cases, a simple keyword matching search is far from enough to satisfy the user's information need. On one hand, natural language is notorious for its semantic ambiguity such as polysemy and synonymy. A word can have multiple meanings and multiple words can also have the same meaning. On the other hand, in most cases, due to knowledge limitation, people don't know how to exactly express their information need via natural language words.

To give a more informative search, this era demands a much more intelligent enterprise search engine that could exploit the deep "knowledge" hidden in the sea of data to help users find their needed information.

SF1-R is an intelligent search engine that does various data-mining analysis to explore the hidden knowledge from the variety of enterprise data to guide the user describe their information need, assist the searching, ranking and presentation of the search result.

Generally, data mining (sometimes called data or knowledge discovery) is the process of analyzing , learning and extracting patterns from data. In enterprise search, the data mainly assumes the textual or multimedia form. By data mining on the massive enterprise data, SF1-R want to gain more perspectives on the

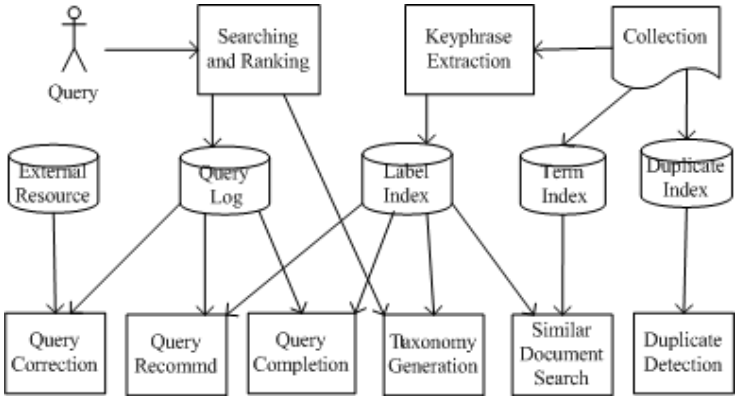

Figure 1.1: The Architecture of Data Mining Algorithms in SF1-revolution.

patterns or knowledge hidden in the collection and summarizing it into structural information to assist the information finding process.

Specifically, in SF1-R's perspective, data mining helps in the following aspects when a user uses the enterprise search engine.

- Help users express their information need through natural language words.
- Provide additional information to assist the searching and ranking against the indexed data.
- Help user locate on their useful information quickly in the search result.
- Eliminate the annoying information such as (duplicate content) in the search result.
- Provide quick overview of the implicit knowledge aspects for the user's query.
- Help users gain more in-depth insights for his interesting topics.

Figure 1.1 illustrates the overall architecture of data mining algorithms in SF1-R. In the following chapters, we will introduce the separate components in detail one by one.

Chapter 2

Keyphrase Extraction

2.1 Background

Keyphrases represents the most salient phrases in the documents that best capture the topics of the documents which provides semantic metadata that summarize and characterize the collection's content.

In SF1-R, keyphrase extraction and indexing lay the basis for many data mining features such as search result clustering, taxonomy generation, query recommendation, query auto-completion, document similarity mearsuring etc.

Keyphrase extraction could be differentiated as document level and collection level ones. The former usually identifies keyphrases using single document based statistics and contextual features around the phrase while the latter extracts keyphrases that is significant in the collection level based on collection level statistics. SF1-R prefers a collection level keyphrase extraction because of the following reasons.

- In a search engine, users seek the needed information against a document collection.
- Keyphrase is used in query recommendation, query auto-completion and other data mining algorithms which surely should be statistical significant in the collection level.

Specifically, the task of keyphrase extraction in SF1-R could be stated as:

Given a collection of documents $C = d_1, d_2, ..., d_n$, extract the set of keyphrases $L = l_1, l_2, ..., l_m$ from C that best captures the content and topics in C.

2.2 Related Works

Keyphrase extraction is one of the main tasks in Natural Language Processing and Data Mining community. Many works have been done in this field. We give a brief overview of the related works so that we could better underatand why we choose the current method in SF1-R.

In document level keyphrase extraction, (1; 2) apply supervised machine learning techniques to classify keyphrases from other noises. Specifically, (1) exploits bayesian classifier and uses mainly TF*IDF score and the distance from first occurrence of the phrase as features; (2) uses decision tree classifier and applies more features such as phrase length and constituent word occuring pattern etc.

In collection level keyphrase extraction, most works use statistical measures to identify those significant phrases in the collection. (3) uses mutual information statistic to discover bigram keyphrases. . (4) proposes a language model approach for key phrase extraction. It uses the pointwise KL-divergence between multiple language models for scoring both phraseness and informativeness of a candidate phrase, which is then unified into a single score to rank extracted phrases. (5) presents a suffix-tree based approach to extract multi-word keyphrases. (6) uses suffix-tree to extract phrases for search result clustering. More recently, (7) uses the DMOZ open directory informatin to rank the candidate phrases used for document clustering.

Comparing those algorithms, the single document based approaches are not suitable for the application purpose in a search engine environment like SF1-R. It is not effective enought to use mutual information alone to select key phrases. Supervised techniques need much labeled data, while the language model based method doesn't consider the topic influence for key phrase extraction. The DMOZ-based algorithm relies too much on exteranl knowledge that might overlook the domain specific information underlying the collection. Also, it is language dependent. It could not be used for languages when there is no similar resources like DMOZ.

2.3 Algorithm in SF1-R

The suffix-based method is very suitable to be used in SF1-R. On one side, based on the full suffix information of text strings in the collection, many sophisticated statical measures could be applied to achieve a good performance. On the other side, although the method is language independent, it provides specical benefits for lanuages like Chinese. In Chinese, there is no word boundaries in the text string. Also, there are no strict differentiation between character, word and phrase. Suffix based methods provides a way to extract significant character strings while need not consider whether it is a word or phrase.

The overall algorithm of using suffix tree for keyphrase extraction could be separated into different phases.

1. Lexical pattern candidate generation.
2. Complete lexical pattern identification.
3. Significant lexical pattern extraction.

2.3.1 Candidate Generation

Generally, statistical methods first generate a set of candidate phrase on which the statistical measures will be applied to evaluate and generate the statistical significant phrases. In ths suffix-based algorithm, first, a set of lexical patterns is generated for the collection which functions as the candidate keyphrases for further statistical selection and filtering.

There are two main approaches in candidate phrase extraction. One is to extract noun phrases. The other is to simply extract continuous n-gram content words. The motivation of the former approach lies in that key phrases are usually noun phrases. However, noun phrase extraction need to do much natural language processing work on the text, such as pos tagging, shallowing parsing, etc., which makes this approach inefficient. So, most works resort to a simple way by extracting continuous n-grams from the text after eliminating stop words. The motivation for this is that key phrases are mainly constituted by content words

while function words (stop words, punctation markers, brackets, numbers, etc.) work as a good boundary indicator for candidate key phrases.

In our suffix-based algorithm, first, text chunks are extracted by function words. Then, all the text chunks are indexed in a suffix-tree like structure. Using such a structure, all the suffix and prefix of the text chunks could be accessed in an very efficient way. After that, all the lexical patterns are generated which are defined as the text strings that consists of more than one successive words (or characters in Chinese) and has certain occurrences in the collection. Those lexical patterns are viewed as the candidates for keyphrases.

2.3.2 Label Selection and Filtering

Based on the suffix information of text string, SF1-R adopts a combination of many statistical measures for keyphrase selection and filtering, including many novel statistical measures like the biased contextual entropy etc. In the following sections, we give brief introduction for those exploited statistical measures.

2.3.2.1 Mutual Information and Loglikelihood

Mutual information has been widely used in measuring the statistical coocurrence and relevance of two terms in natural language processing. It is defined as:

$$MI(x,y) = \frac{f(x,y)}{f(x)f(y)} \tag{2.1}$$

in which, f(x,y) is the coocurring frequency of term x and y while $f(x)$ and $f(y)$ is the frequency that term x and term y occurs in the collection respectively.

It is well known that mutual information will give false high scores when the terms occur rare in the collection. To compensate this problem, a second measure, log-likelihood ratio, is used. logL is more robust to low frequency events. Let $f(x,y)$ be the frequency of two terms, x and y, occurring adjacent in some corpus (where the asterix (*) represents a wildcard). Then, the log- likelihood ratio of x and y is defined as:

$$logL(x,y) = ll(\frac{k_1}{n_1}, k_1, n_1) + ll(\frac{k_2}{n_2}, k_2, n_2) - ll(\frac{k_1+k_2}{n_1+n_2}, k_1, n_1) - ll(\frac{k_1+k_2}{n_1+n_2}, k_2, n_2) \tag{2.2}$$

where $k_1 = f(x, y)$, $n_1 = f(x, *)$, $k_2 = f(\neg x, y)$, $n_2 = f(\neg x, *)$ and

$$ll(p, k, n) = klog(p) + (n - k)log(1 - p) \tag{2.3}$$

The shortcoming of loglikelihood is that the value will be also high for two frequent terms that are rarely adjacent. For example, the word pair (the, the) has a very high log-likelihood ratio in English even though it rarely coocurrs sequentially. Mutual information and log likelihood ratio could complement each other. So, a hybrid metric based on the combination of mutual information and loglikelihood ratio is used, as shown below.

$$Sig(x, y) = \begin{cases} logL(x, y) & \text{if } MI(x, y) \geqslant \alpha \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

Mutual information and Loglikelihood are traditionally used for measuring the statistical significance of two terms. For lexical patterns with a length larger than two, we extend them to the following measures. Suppose that $x = x_1, x_2, ..., x_n$ is a candidate lexical pattern with multi-terms. Then,

$$MI(x_1, x_2, ..., x_n) = MI(f_l, f_r) \tag{2.5}$$

where $f_l = x_1, ..., x_{n-1}$ and $f_r = x_2, ..., x_n$ are the two longest substrings of x.

$$\begin{array}{l} logL(x) = logL_1(x) + logL_2(x) + logL_3(x) \\ logL_1(x_1, x_2, ..., x_n) = logL(f_l, f_r) \\ logL_2(x_1, x_2, ..., x_n) = logL(f_l, x_n) \\ logL_3(x_1, x_2, ..., x_n) = logL(x_1, f_r) \end{array} \tag{2.6}$$

2.3.2.2 Biased Contextual Entropy

Candidate phrases with high score of mutual information and loglikelihood doesn't necessarilly mean the keyness of those candidates. There are two main cases of false positives. First, some phrases are incomplete. For example, "chain monte" is not a keyphrase because it is only part of a complete phrase of "Markov Chain Monte Carlo". Second, some phrases are very common phrases across domains, for example "every day".

To deal with the first problem, the weighted contextual entropy is used to measure whether a candidate is a complete phrase or not. The basic idea is

that if a candidate is a complete lexical pattern, then its adjacent words on left and right will have greater variety then those incomplete lexical patterns. For example, for the incomplete candidate "chain monte", the possible left adjacent words occur in the collection maybe only "Monte", while there maybe a lot of different words appearing in the left position of the complete lexical pattern "Markov chain monte carlo".

The contextual entropy is defined as follows:

$$CE(x) = -\sum_{t \in C} \frac{f(t)}{TF} log \frac{f(t)}{TF} \tag{2.7}$$

where C is the set of contextual terms of the candidate x. $TF = \sum_{t \in C} f(t)$ is the overall context occurrence frequency. CE indicates the variety of the contectual terms assocaited with the candidate phrase. However, as we stated earlier, there is a great variety of words that could possibly occur in the context. Also, besides content words, there are also possible functional delimiters that appears in the left or right contextual positions of the candidate. Different groups of words have different weight for the contribution of the contextual entropy. For example, the occurrence of different numbers like "one", "two" maybe not as strong as the different occurrence of other content words for indicating the completeness of a lexical pattern. So, special bias or weighting should be considered for those delimiters which may indicate good or bad candidates in different cases.

Motivated by the above observation, we define a biased contextual entropy:

$$BCE(x) = -\sum_{g \in C} \frac{w_g * f(g)}{TF} log \frac{f(g)}{TF} \tag{2.8}$$

where g represents the term group occurred in the context while w_g is the weight for the term group. Currently, in the implementation, we differentiate several term groups like "number", different functional words as propositions etc and weight such groups in an empirical way. Actually, the model could be developed to be very powerful by incorporating some supervised mechanism to train g and w_g which will be explored in the next section.

2.3.3 Automatic Training of the BCE Model [To do]

A referentical corpus R could be used to train and optimize the parameters in the BCE model automatically.

2.3.4 Keyphrase Filtering [To do]

Using the statistical measures alone could still not determine the keyness of the candidate well. For example,common phrases is notoriously difficult to be eliminated. For this problem, we attemp whether some referential domain structure from external source or computed from the collection could help. Specifically, we try to model a candidate phrase's significance by considering its relative contribution with respect to a set of different (domains) topics. The underlying intuition lies in that a keyphrase should represent the typical content for a specific domain or topic. In other words, it should focus on some aspect of a specific topic. We call this property the Keyness property. In the following, we define the keyness as the candidate's topical distribution entropy.

Formally, suppose there is a corpus C and the set of domains (or topics) $T = t_1, t_2, ..., t_k$ in C. Let $L = l_1, l_2, ..., l_m$ be the set of candidate phrases extracted from C. Let $L^* \subset L$ be the target keyphrase set.

$$keyness(l) = \sum_t p(t|l) log p(t|l) \tag{2.9}$$

where

$$p(t|l) \propto p(l|t)p(t) \tag{2.10}$$

Given the candidate phrase $l = w_1 w_2 ... w_n$, $P(l|t)$, the generation probability by the topic. Assuming a bigram language model, we have

$$p(l|t) = p(w_1|t) \prod_{j=2...n} P(w_j|w_{j-1}, t) \tag{2.11}$$

In the model, note that we don't make specific requirement for what a topic (or domain) should be. Actually, it could be those explicit categories such as in open direcotry project (DMOZ) or subjects in semi-structured knowledge bases such as Wikipedia. Or ideally, it could be those different category of documents produced from a document classification algorithm on the collection.

Chapter 3

Taxonomy Generation

3.1 Background

Search engines help users to meet their information need. When users enter queries to the search engine system, oftentimes, however, they do not know how to represent their information need precisely. So, in those situations, users tend to input an unspecified query to the search engine.

One of the design goal of SF1 is to provide not only the documents for the unspecified query, but also a set of choices and navigation points, so that users can narrow down their searches to land on the documents they want. The taxonomy generation component is designed to meet the above requirement. It should provide a taxonomy structure for the search result as a supplement for the initial ranking list.

We must first make a choice as what kind of taxonomy structure we want to show in the result. When a user forming a query, it represents a underlying concept that the user want to know about. The more unspecified the query, the more general the concept underlying the query. A concept usually has many closely related facets (attributes) with it. For example, if a user inputs a query "NBA", he may want to know something about the "National Basketball Association". This concept has many attributes with it, for example, "teams", "players", "playoffs" and "history" etc. By searching the query in a collection of documents, it will return a set of documents that all talk about "NBA". However, different documents may talk about different facets of this concept. We could group the search

results by their topic on different facets of the concept. The grouping could be recursive. For example, the concept "NBA teams" may also have its own facets. So, it could form a taxonomy structure, in the top of which is the unspecified user query. We call such a taxonomy structure as query-centered taxonomy.

We should also make clear what properties we want for the query-centered taxonomy. Generally speaking, a proper taxonomy structure should be effective to provide enough information to help users to find their needed documents. At the same time, it should be able to be computed in a very efficient way. Ideally, we think it should have the following properties.

1. It should cluster the top results of the original query into a hierachy of folders which are labeled with variable-length sentences.

2. The labels of the folders should capture the "theme" of the documents in the folder.

3. The labels in the taxonomy should be closely related to the original query.

4. A sub-node should express more specific concepts than its parent node.

5. The sibling nodes should different each other and represent parallel concepts.

In the following part of this report, we will first introduce the background and related works of the current research in Section 3.2. Then, in Section 3.3, we will introduce architecture design of the taxonomy generation coponent in SF1-revolution. After that, we will dive into the details of the algorithm selections for the implementation. Then, we will evaluate the different implemented algorithms. We finally conclude the work in the end of the document.

3.2 Related Works

Basically, there are two approaches to cluster the search result and generate taxonomy navigation structures. One is to first cluster the documents and then label the clusters. The other is to first generate labels that represent the documents'

content and then cluster the documents with respect to the set of labels. We call the former cluster-based approach and the latter label-based approach.

3.2.1 Cluster-based Taxonomy Generation

Given a collection of documents, traditional clustering could be used to orgnize the documents into a hierarchy. The intuition of those approaches is the cluster hypothesis which states that relevant documents to a specific topic tend to appear in the same cluster. We could use partional clustering algorithms like K-means to iteratively cluster a set of documents into hierachical clusters. To generate more fine-grained taxonomy structure, we could also use agglomerative clustering algorithms. Agglomerative algorithms begin with each element as a separate cluster and merge them into successively larger clusters to form fined-grained hierachical structure.

Those clustering algorithms all rely on similarity computation between representations based on the whole document content. For example, the most popular way is to represent the document as TF*IDF vector based on bag-of-word assumption and compute the similarity between documents by Cosine measure between the document vectors. SCATTER/GATHER (8) is one of the first web-clustering softwares on top of an IR engine which exploits the Fractionation algorithm to organize search results into thematic groups. (9) uses transactional K-Means while (10) uses K-centers algorithm to produce the flat clustering of Web search results.

To enable the user to quickly identify and understand the topics of the document clusters in the taxonomy, the clusters should be labeled with short natural language sentences that could capture the themes of the clusters. (10) first selects candidate words for each cluster by means of a modified version of the Information Gain measure. Then, the shortest substring of a document title among those having the highest score with respect to the candidate words of each cluster is selected as the label of the cluster. (11) casts the labeling problem as an optimization problem involving minimizing Kullback-Leibler divergence between word distributions and maximizing mutual information between a label and a cluster represented as multinomial distributions.

The cluster-based algorithm could be done statically or dynamically. In the first case, the whole collection under retrieved is clustered before seeing the query. In the second case, only the top ranking results are clustered for a specific query. Because the user query is relatively open and the matching of the query to a particular part of a static taxonomy is difficult, the static approach could not reflect the user's information need well. In the dynamic approach , instead of the whole collection, the top part of the returned document set of a query is dynamically clustered and labeled. However, because the cluster-based algorithms cluster documents based on the whole document's representation. The algorithms suffers from efficiency issue. Also, even in the dynamic approach, because the cluster labels is usally assigned based only on the document content in the cluster, the taxonomy generated could still not reflect the original user query well.

3.2.2 Label-based Taxonomy Generation

From the above section, we see that the cluster-based taxonomy generation methods are not suitable to generate a query-centered taxonomy we want as introduced in Section 3.1. Instead, label-based methods are more suitable for this task. Different from cluster-based methods, label-based methods first extract from the research results of a query the set of salient labels (words, phrases or short sentences) that best represents the content of the document set. Then, documents are grouped by their similarity with respect to whether covering those salient labels.

Grouper (6; 12) makes the first attempt in using recurring phrases as the basis for deriving similarity of documents. It dynamically groups the search results into clusters labeled by phrases extracted from the snippets. First, it creates an inverted index of phrases of the document set using a suffix tree. Then, it clusters the documents by their share of the phrases in the document collection. Lingo (13) uses a combination of phrases and Singular Value Decomposition (SVD) techniques for clustering. First, in Lingo, frequent terms and phrases are discovered in a combined set of all documents by smart utilization of suffix-arrays. Then, SVD is used to extract orthogonal vectors of the term-document matrix, believed to represent distinct topic in the input data. A description of each orthogonal

vector is assembled from previously extracted common phrases by using a Vector Space Model and calculating score of each phrase against the SVD-decomposed matrix. Finally, the algorithm applies group labels as artificial queries against the input document set. Highest scoring documents for each cluster are assigned as that cluster's content. (14) reports a system that first extract phrases using suffix tree and then uses supervised regression to identify salient phrases as labels. Based on the labels, the documents are clustered. The above algorithms all create flat clustering structure.

Compared to flat structure, hierarchical taxonomy view is more attractive. SHOC (15) exploits a variation of suffix array for key phrase extraction and origizes the folders into a hierachy. They use the SVD decomposition on the label document matrix to clustering documents and then merge the clusters into hierachy in a bottom-up manner. (16) first builds off-line a statistical model of the background language and subsequently extracts "topical terms" from the documents using measures for "topicality" and "predictiveness. Then, it uses a recursive algorithm to create the hierachical structure of document folders. DisCover (17) uses general noun phrases and single nouns and adjactives as the candidate labels and propose a heuristic score function that combines the "document coverage" and "sibling node distinctiveness" properties to select the most salient labels and clusters the documents containing the labels into hierachical folders. SNAKET (7) uses the DMOZ open directory informatin as a knowledge base to extract gapped sentences of variable length as labels. Then, it uses a bottom-up method operated on the set of labels to construct the hierachical folder structure.

As shown above, compared to cluster-based method, on one side, the label-based method is more efficient in clustering the search result into hierachical folders. On the other side, more importantly, the label-based method could better reflect the query-centered view of taxonomy. So, in the algorithm design of taxonomy generatin in SF1-R, we adopt the label-based view. In the following, we will dive into the details of our architecture and algorithm design in SF1-R.

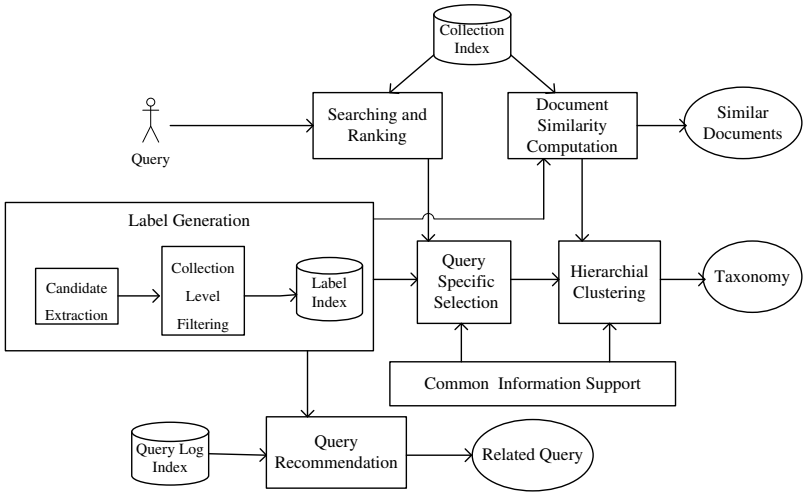

Figure 3.1: The Architecture of Taxonomy Generation in SF1-revolution.

3.3 Architecture Design

Figure 3.1 shows the overall architecture design of the taxonomy generation in SF1.

3.4 Static Label Indexing

3.4.1 Keyphrase Extraction

The labels used to name a folder in the taxonomy can be single words, phrases or short sentences. Among those choices, phrase is the most suitable one. The keyphrase extraction module in TG relies on the keyphrase extraction algorithm as introduced in Chapter 2 which extracts the salient multi-word concepts that represent the main topics and content in the whole collection.

3.4.2 Document Specific Filtering

Given a collection, we want to index for each document its associated labels which will be used in online query time to cluster the search result and generate the hierarchical navigation information. For each document, the appearing keyphrases

extracted will be viewed as the candidate label set to represent the document's main content and topic with respect to the whole collection.

A document specific filtering is further done to select the candidate labels to index for each document. A document may have many labels apearing in it and some labels may not as important as others in representing the document specific content. Also, the number of labels in a document has direct influence on the computation speed for online taxonomy generation. So, a document specific filtering is required to prune those insignificant labels with respect to the document before indexing.

For filtering the label candidate in a document, we must have a way to measure their contribution to the document specific content. A naive way is to use the standard TF*IDF formula. However, the standard TF*IDF relies too much on the IDF weighting. For example, in a large collection, when a label's document frequency is very small, TF*IDF will give a very high score to the label. However, one of our main purpose of using labels in TG is to cluster the search result. It is well known that clustering sometimes prefer a high document frequency score.

Given the above reasons, through experiments, we propose a smoothed version of TF*IDF scoring for document specific label weighting as follows:

$$W_d(l) = sqrt(f_d(l) + 1) * log(|C|/(df(l) + s) + 1) \tag{3.1}$$

where l is the candidate label; $f_d(l)$ means the label's occurrence frequency in document d; $|C|$ denotes the total number of documents in the collection C while $df(l)$ represents the document frequency of label l; s is a smoothing factor computed as:

$$s = \begin{cases} \frac{|C|}{\alpha} & \text{if } \frac{|C|}{\alpha} < \beta \\ \beta & \text{otherwise} \end{cases} \tag{3.2}$$

Using the above label weighting method, the top M labels with the highest document specific weighting score is selected to be indexed.

3.4.3 Compression and Storage

For each document, there is a list of labels that represents the document's topic with respect to the collection. When the document size becomes very large,

the index will easily exceed the memory capacity for a general server used for enterprise search. So, the indexing should support second storage on disk. At the same time, it should support very efficient querying. We use a file-based data structure with a good memory caching method together with an effective index compression strategy for storing the label index.

For the compression of the label list associated with a document, we use the variable length method which is popular and effective for compressing posting lists in information retrieval. Specifically, first, the labels are transformed to incremental IDs starting from 1, with frequent labels receiving small integers. Then, the posting of the label list for a document is sorted according to their IDs in ascending order. After that, the IDs are replaced by their gaps with the previous ID. On such variable length data, the D-Gap compression is applied to make a compact storage.

3.5 Online Taxonomy Generation

3.5.1 Query Specific Filtering

Given the static label index as introduced in the above section, When a query comes, we use the labels from the top ranked n document list to generate the taxonomy. Before generating the taxonomy using hierarchical clustering algorithm, we should first filter the insignificant labels with respect to the query. Specifically, we want to select a subset labels that are most related to the query to form the query-centered taxonomy.

Currently, we attempt to use two strategies for this task, Minimum Document Support and Label Generality with respect to the query. Document support means the number of documents in the top search results that contains the label. Label Generality represents the label's conceptual generality with respect to our the concepts underlying the collection.

First, we think that related labels should co-occur with the query words in the top relevant documents of the query. Suppose R_K represents the top K ranked results with respect to query q. Let $supp_l \in R_K$ be those documents that contain

the label l. The Minimum Document Support criterion selects those labels that satisfy $minsupp \leq supp_l \leq maxsupp$ ($1 \leq minsupp \leq maxsupp \leq N$).

Second, the labels with high document support in the top searching result doesn't necessarily mean that it must be a very good label to appear in the taxonomy result. First, some of which may be very common concepts with respect to the whole collection that provides no more specific information related to the information need underlying the user query. Second, the generality of the labels selected should be in the same level with the user query. For example, if a user inputs a query that represents a very general concept with respect to the collection, then the labels in the taxonomy should be also general concepts. On the other side, if the users inputs a very specific concepts with respect to the collection, then the labels appeared in the taxonomy should be also specific concepts in the similar level with the query. Given the above reasons, a query specific label generality measure is used to filter our labels that are either too general or too specifc compared with the concept underlying the user query.

Let q be the user query. l is the label measured. C represents the whole collection. The Label Generality of l with respect to q is defined as:

$$Generality_q(l) = \frac{log(supp(l) + 1)}{log((df(l) + s)/log(sqrt(ret(q)) + 1) + 1)} \tag{3.3}$$

where $supp$ represents the number of documents that contains l in the top K documents from q's search result against C; $df(l)$ is the document frequency of l in the whole collection C; $ret(q)$ means the number of returned result of q when searching against C. s is a smoothing factor defined in the same way as in formula (3.2).

3.5.2 Hierarchical Clustering

3.5.2.1 The Algorithm

Besides label generation and selection, hierarchical clustering based on those labels plays another core role in taxonomy generation. As we stated in Section 3.1, we require the taxonomy generated to have several properties. While label generation and selection enable the labels to reflect the theme of the document

and relatedness to the original query. Hierarchical clustering should organize the documents into hierarchical folders, such that sub-nodes represent more specific concepts than their parent node and sibling nodes express parallel concepts. Specifically, we should aim to maximize the following two properties between labels in the clustered result.

- (Fathership) A sub-node should express more specific concepts than its parent node.
- (Sibship) The sibling nodes should differentiate each other and represent parallel concepts.

In traditional hierarchical clustering, there are two popular ways to cluster a set of documents into hierarchical taxonomies. One is top-down and the other is bottom-up. Most previous works in taxonomy generation adopt a bottom-up method for hierarchical clustering based on labels (7; 15). In those works, labels with similar supported document set are recursively merged to form the hierarchy. Different from those, (18) proposes the Discover algorithm which works in a kind of top-down manner. It selects the sibling nodes under a specific node according to an intuitive function that measures the coverage and distinctiveness of the nodes.

The previous works have some weakness to meet our expectation of taxonomy generation. First, they usually only consider one aspect of the taxonomy. For example, (7; 15) only considere sub-node and parent-node relation while (18) only considers sibling relation. Second, previous works measures the relations between labels using only the support set of labels and don't consider the semantic relatedness between labels.

To meet our expectation, we propose a clustering algorithm by considering explicitly the relations between parent node and its child nodes as well as relations between sibling nodes. Our clustering algorithm has an additional feature of considering the subcategory head of query words and conceptual relation between labels. Specifically, previous works measures the relations between labels using only the label's support set. In our algorithm, we also consider semantic relatedness between labels by considering syntatic patterns in a simple while efficient way.

To well describe our algorithm, we first define some notions more formally in the following.

Definition 1 (Label Support Set). Let $l \in L$ be a sleeted label for query q. Let $D = \{d_1, d_2, ..., d_m\}$ be the top returned document set of the query q. Then, the support set of l is $supp(l) = \{d_{l_1}, ..., d_{l_n}\}$ where $l_i \in 1...m$ and the label l appears in document d_{l_i}.

Definition 2 (Admissible Taxonomies). A taxonomy could be formally described as 3-tuple $h = (L, R, W)$, where L is the set of node and $R \subset C \times C$ is an partial order on L which denotes the set of taxonomy inclusion relations between nodes, and $f_w \subset R \times \Re$ is a function that weight each pair in R_i with a real number. Further, we define the admissible taxonomy set with respect to a label set L to be all the possible taxonomies possibly produced over L. That is, $H = \{h_1, h_2, ..., h_m\}$.

Definition 3 (Optimal Taxonomy). First, for each admissible taxonomy h, a scoring function f is defined as $f = \Sigma_{i=0}^{|R|} f_w(R_i)$. Then, the optimal taxonomy with respect to q is defined as the admissible taxonomy with the maximum score.

$$h^* = \arg\max_i f(h_i) \tag{3.4}$$

Our goal is to compute the optimal taxonomy with respect to the label set produced by the query q. However, it is too complex to resort to the global optimum. Actually, we could resort to some approximation methods. In algorithmic level, we could start by assuming the taxonomy h only contains an node q which denotes the original query. Each time, we compute the maximum scoring pair $f(l_i, l_j)$ between nodes $l_i \in h$ and the remaining labels $l_j \in L'$ not added to taxonomy. Then we add the node l_j to h under l_i.

3.5.2.2 Measuring Fathership and Sibship

From the above description, we can see that the most important factor in the algorithm is to define the weighting function $f_w(R_i)$ where $R_i = (l_{i_1}, l_{i_2})$ is a possible inclusion relation in the taxonomy. We define it explicitly by the Fathership and Sibship between labels by exploiting a combination of the label's support document set and syntactic relatedness.

- *Fathership*

$$
\begin{aligned}
&F(l_i, l_j) = F_{supp}(l_i, l_j) + \alpha_1 * F_{syn}(l_i, l_j) + \alpha_2 * F_{pat}(l_i, l_j) \\
&F_{supp}(l_i, l_j) = \tfrac{|supp(l_i) \cap supp(l_j)|}{|supp(l_j)|} \\
&F_{syn}(l_i, l_j) = \begin{cases} \tfrac{|words(l_i) \cap words(l_j)|}{|words(l_j)|} & \text{if } |words(l_j)| >= |words(l_i)| \\ 0 \quad \text{otherwise} \end{cases} \\
&F_{pat}(l_i, l_j) = freq(last(l_j), P_F, first(l_i))
\end{aligned} \tag{3.5}
$$

- *Sibship*

$$
\begin{aligned}
&S(l_i, l_j) = S_{supp}(l_i, l_j) + \beta_1 * S_{syn}(l_i, l_j) + \beta_2 * S_{pat}(l_i, l_j) \\
&S_{supp}(l_i, l_j) = |(supp(l_i) - \cup_k supp(child_k(l_i))) \cap supp(l_j)| \\
&S_{syn}(l_i, l_j) = \textstyle\sum_k \tfrac{|words(child_k(l_i)) \cap words(l_j)|}{|words(l_j)|} \\
&S_{pat}(l_i, l_j) = freq(last(l_j), P_S, first(l_i))
\end{aligned} \tag{3.6}
$$

where $words(l)$ represents the set of words occur in l, $first(l)$ means the first word in l and $last(l)$ means the last word in l. P_F and P_S is the set of indicating words for *Fathership* and *SibShip* respectively. For example, in English, we define $P_F = \{of\}$, $P_S = \{and\}$. In such a case, $freq(w_i, of, w_j)$ means the number of times the lexical pattern "w_i of w_j" occur in the collection, while $freq(w_i, and, w_j)$ means the number of times the lexical pattern "w_i and w_j" or "w_j and w_i" occur in the collection This information is statically computed for the overall collection.

The overall scoring function is then defined as the combination of labels' fathership and sibship score:

$$f(l_i, l_j) = F(l_i, l_j) * S(l_i, l_j) \tag{3.7}$$

3.5.3 Post-Processing

3.5.3.1 Label Deduplication

Many labels are possibly duplicates for each other and provide redundant information which will annoy the users very much if those duplicates appear in the taxonomy result. For example, in English, many labels are just morphology or case variations to each other. In Chinese, many duplciates also exist such as synonymy variations, abbreviation variations etc.

The label deduplication could be done offline in index time or online in query time. We prefer an online label deduplication because of the following reasons.

1. Those labels are not identical to each other. Many duplicates are actually some semantic variations.

2. Specific form of variations is salient and appropriate only with respect to certain contexts. It is difficult and inappropriate to only retain a standard "form" in the collection level with all other "duplicate" labels eliminated in offline time. Actually, without the specific query and result context, sometimes we could not determine which form should be the standard form.

3. The labels generated online is relatively small. The cost for label deduplication in online time is negligible.

We create many filters for the online label deduplication task, such as abbreviation filter, single letter phrase filter, wholly-contained subphrase filter, morphology variation filter etc. Note that many filters for label deduplication are language dependent.

3.5.3.2 Grouping Insignificant Nodes

Under a parent node, there may be several child nodes whose support set is very small in the returned set. The existence of many of such nodes may become the nuisance for the users. So, under each parent node, the set of insignificant child nodes are collapsed into a folder named "Others". The conditions for identifying l as an insignificant nodes is:

- $|words(l) \cap words(q)| = 0$
- $supp(l) = 1$

3.5.3.3 Identification of Subcategory Head [To do]

Some post-processing operations should be further done to improve the taxonomy's naturalness. First, we introduce the notion of subcategory head. Given the

original query q , we say that a label l is a subcategory head of q if it expresses one of q's attribute with the minimum words.

The introduction of this notion is to make the taxonomy become more natural. To reduce the complexity and increase accuracy, in the label generation phase, we don't extract one word labels. However, in some cases, especially in the first level of the taxonomy for short queries, it is natural to add some one word labels there. For example, if the query is "NBA", then both "teams" and "Junior NBA" could be viewed as subcategory head of "NBA" while "NBA teams" is not. For example, people don't want to see "NBA teams", "NBA players", "NBA scores" to appear under the node "NBA". Rather it is more natural to list as "players", "teams", "scores" under the node "NBA" because there is no ambiguity in such a context.

Under the above motivation, for short queries, we identify the subcategory heads in the first level of the taxonomy. We use the syntatic occurrence patterns for the identification. Specifically, a word w will be identified as a subcategory head if all the following conditions are satisfied.

- w occurs in label $l = qw$ where l represents a subnode under the query root.
- $freq(last(q), w) > t_1$
- $freq(w, P_F, first(q)) > t_2$;

Chapter 4

Similar Document Search

4.1 Background

When a user reads a document in the search result, he may become very interested in the topic of the document and want more documents with similar topic or semantic content. Similar Document Search is an effective feature in SF1-R to satisfy the user's inforamtion need considering the above scenario. The users can search the document at hand against the whole collection to get the most similar documents in the same topic.

Documents could be similar with each other in two different ways, syntactic or semantic. In the former case, the two documents may have nearly duplicate content or content segments in text writing. So, one document could actually be viewed as a redundant one of the other. In the latter case, the two documents discuss the similar semantic topic while they are not necessary syntactic similar documents. Actually, SF1-R views the similar documents to be those semantic similar documents with duplicate or near duplicate documents eliminated.

The semantic similarity between documents is a basic concept in Natural Language Processing and Information Retrieval. There are many similarity measures between documents such as the well known TF*IDF measure based on the Vector Space Model (19), the KL divergence measure based on the Language Model (20).

No matter what kind of similarity measure used, an online computation of the similar documents for a query document is time-costing and infeasible. Typically,

given a collection of documents, a static similarity index should be built offline, so that each document's similar document list with high similarity score will be pre-computed. The problem could be characterised as All Pairs Similarity Search problem that requires finding all pairs of records with similarity scores above a specified threshold. We specify the problem with regard to application scenario in SF1-R as follows:

> Given a set of documents $C = d_1, d_2, ..., d_n$, a similarity measure function $sim(x, y)$, and a similarity threshold t, build the index in which a given document d_k is associated with the set of documents $K = d_{k1}, d_{k2}, ..., d_{km}$, such that $\forall_i \{d_{ki} \in K \rightarrow sim(d_k, d_{ki}) > t\}$ and $\forall_{i,j} \{i > j \rightarrow sim(d_k, d_{ki}) > sim(d_k, d_{kj})\}$.

The similarity indexing problem is an $O(N^2)$ problem. When N becomes large, a naive solution by pair-wise similarity computation is beyond the computation ability of a typical server used in an enterprise search engine. It is also not very suitable to rely mainly on parallel computation such as map-reduce (21) for such problem in enterprise search engines because of the typical resource limit in enterprise search scenario. So, this presents a big challenge in SF1-R.

Faced with the above computation challenge, we apply the index pruning method that is commonly used in search engines to prune the size of term index. At the same time, the effectiveness of the algorithm is guaranteed by multi-field combination and the utilization of key concepts extracted from the document collection.

4.2 Related Works

There are many works that are closely related with our problem at hand. We give an overview to them to better motivate and illustrate our current method implemented in SF1-R.

4.2.1 Approximation on Similarity Measures

Syntactic similar document search has wide application in fields such as duplicate document elimination, plagiarism detection etc. In such works, the approximation

on the similarity measure on a compact document representation is often exploited to reduce the problem complexity. Usually, documents are transformed to some kind of compact representation in bitstring. Then, the similarity between the original documents is transformed to the similarity computation between those compact bitstrings. (22; 23) use shingles which are consecutive overlapping text fragments and Rabin's fingerprint to represent the document compactly. It could be viewed as transforming the well known Jaccard similarity measure to bit-wise similarity. (24; 25) apply locality sensitive hash to transform the documents into compact representations that preserves the cosine similarity.

Those approximate similarity methods have been shown to be very effective in duplicate document detection. However, there are two points that limit its direct application to the semantic similarity search problem. First, by the transformation to the compact representation, much information is lost and the approximate similarity measure becomes very coarse-grained compared with the original similarity measure. Second, even with the compact bitstring representation, a direct implementation of pair-wise comparison is still prohibitive considering efficiency. Usually, some pre-clustering methods on those bitstrings based on the common segments are done to give a fast computation. The effectiveness of such pre-clustering has correlation with the similarity threshold t. A smaller t may lead to very inefficient pre-clustering method. While syntactic similarity detection often uses very big t, the value used in semantic similarity detection is much smaller. Those limits may actually counteract the benefit obtained by applying such methods in semantic similarity computation.

4.2.2 Similarity Computation based on Inverted Index

The semantic similarity computation is closely related with works of relevance evaluation in information retrieval (IR). Different IR models use different document representation and similarity metrics to measure the relevance (similarity) between a user query and the documents in the collection. Commonly, inverted index is used to store the set of document representations in a collection. The similarity between a query string and the documents is then evaluated based on the inverted index.

There are two different approaches in query evaluation against the inverted index (26), document-at-a-time (DAAT) and term-at-a-time (TAAT). The DAAT strategy evaluates the whole query with respect to a single document before moving to the next document. The TAAT strategy processes terms in query one by one and uses accumulator to accumulate each term's contributed partial similarity scores as each term is processed.

To reduce the storage size and computation complexity, pruning is often used in IR to optimize the similarity evaluation process. An important pruning method for DAAT strategy is termed as *max-score* (26) that operates by keeping track of the top k scoring documents seen so far. Evaluation of a particular document is terminated as soon as it is clear that this document will not be ranked in the final top k result. For TAAT pruning, (27), the whole score accumulation of an insignificant term could be saved.

The similarity index problem could actually be done by exploiting the term-at-a-time query evaluation and index pruning idea based on inverted index. First, a term-document inverted index could be built for the document collection. Then, index pruning could be done by eliminating the insignificant terms as well as the unimportant documents in a term's posting list. After that, score accumulation on document pairs could be done by scanning the terms in inverted index in some order. Specifically, in SF1-R, to maximize the effectiveness even with a large pruning factor, we also innovate by exploiting multi-field representation and applying the keyphrases extracted from the documents into such an inverted index based algorithm. In the following sections, we will dive into the details of similarity index and search in SF1-R.

4.3 Algorithm in SF1-R

4.3.1 Document Representation

The representation of a document is the basis for measuring document similarity. Traditionaly in text processing and IR, a document is represented as a bag of words. Numerious methods have been attempted to improve over such a representation. In our algorithm, to make the algorithm computationally feasible,

we use index pruning technology to prune a large part of informtion from the document collection. To maximize the effectiveness on such pruned data, we use two additional resources to complement the traditional document representation. First, we exploit document fields for a semi-structured document representation. Second, we use keyphrases together with the simple terms for document representation.

4.3.1.1 Field Combination

The documents to be indexed often have some kind of structured form. For example, documents are usually orgnized by different fields, such as "title", "author" and "content" etc. In enterpise search scenario, document fields are especially important to represent different aspects of a document item. For example, a document may represent a book item in an online book store.

Previous works in information retrieval has found it to be beneficial to exploit the document's structural information to improve the IR performance. In such approaches, given a query and a document, the relevance of different document fields are evaluated against the query. Then, the overall relevance could be computed as a weighed combination of those field scores (28; 29). The experiments in IR has shown that field score combination boost the performance of relevance evaluation between query and documents.

There are no previous works that use field score combination in the All Pairs Similarity Search problem. We apply it in our similarity indexing algorithm. First, we could benefit from the performance boosting in similarity evaluation as shown in previous works in IR. Second, we construct field based pruning algorithm upon it to maximize the efficiency while maintaining the most important information during term and document pruning. Intuitively, we want to prune more information on those unimportant fields than in the important fields.

4.3.1.2 Exploiting Keyphrases

Given a collection, the keyphrases extracted by the algorithm in Chapter 2 give concise description and index of those salient concepts in the document collec-

tion. We use the keyphrases as a complement to the simple words in document representation.

First, phrases could be seen as some way to capture the relationship between simple words. It has long history of exploiting phrases in document representaion in IR to improve query relevance measures (30; 31). Second, like the usage of field combination, we want to maintain the most important information as much as possible in index pruning. Actually, we could view the keyphrases from a document as forming a pseudo document field. It could be processed with other true document properties in a uniform way. Then, the field-based pruning algorithm could better use the field specific information to attain an optimized trade-off between effectiveness and efficiency.

4.3.2 Similarity Measure

Cosine similarity together with TF*IDF term weighting has been widely used in the similarity computation between document vectors. The commonly used TF*IDF term weighting formula has an intuitive form as :

$$w(t) = TF(t) * log(\frac{N}{DF(t)} + 1) \tag{4.1}$$

where $TF(t)$ means the term frequency of t; $DF(t)$ represents the document frequency of t in the collection while N represents the total number of documents in the collection.

The cosine similarity based on TF*IDF doesn't perform well in our similarity indexing problem. As we know, cosine similarity with the intuitive TF*IDF has bad length normalization. In IR, it has been shown that it may give false high score to very short documents. When applying it in our multi-field score accumulation algorithm, the above shortcoming presents big difficulty for the combination of scores from different document fields.

Through experiments, we find the term weighting method used in the probabilistic IR model Okapi BM25 (32) performs very well in the similarity index problem. The term weighting in BM25 could actually be seen as an TF*IDF variation by length normalization with solid matchematical background.

$$w(t) = TF\_LN(t) * IDF\_LN(t) \tag{4.2}$$

$$TF\_LN(t) = \frac{(k_1 + 1) * TF(t)}{k_1((1 - b) + b\frac{|d|}{avdl}) + TF(t)} \tag{4.3}$$

$$IDF\_LN(t) = ln\frac{N - DF(t) + 0.5}{DF(t) + 0.5} \tag{4.4}$$

where k_1 and b are parameters; $|d|$ is the length of the document while $avdl$ is the average document length in the document collection.

For two documents d_1 and d_2, their similarity based on the BM25 term weighting scheme is then computed as:

$$Sim(d_1, d_2) = \sum_{t \in d_1 \cap d_2} (w_{d_1}(t) * w_{d_2}(t)) \tag{4.5}$$

In the formula, note that the overall similarity between the two documents could be viewed as the addition of all term weights shared by the two documents. This provides a basic form to accumulate the similarity scores between document pairs in the term-at-time way based on the inverted index.

In our multi-field cases, the similarity is actually computed between the corresponding properties of the documents. Then, the similarity score from multiple properties are combined to form the document similarity score. The measure on multi-field documents is shown as:

$$Sim(d_1, d_2) = \sum_{p \in P} (w_p * Sim(p_{d_1}, p_{d_2})) \tag{4.6}$$

where P is the set of properties the documents share in the collection. w_p is the combination weight of property p.

4.3.3 Score Accumulation based on Inverted Index

As stated in the above section, the simlarity score measure used enable us to accumulate scores based on the inverted index. In the multi-field inverted index, for each term t and property p, there will be a posting list $posting_{t,p}$ corresponding to all the documents that have term t appeared in p. We define the similar document pair candidates on $posting_{t,p}$ as:

$$Cand_{t,p} = \{(d_i, d_j) | d_i, d_j \in posting_{t,p}; d_i \neq d_j\} \tag{4.7}$$

On each candidate document pair (d_i, d_j), its term weight is computed as $w_p * w_{d_i,p}(t) * w_{d_j,p}(t)$. For all the possible similar document pairs, their scores are accumulated when the pairs occur in the candidate set of the terms scanned. After all the terms in lexicon T of the inverted index are scanned for all properties, we obtain the final document similarity score for each candidate similarity pair according to Formula (4.6). Then, all the candidate document pairs with an accumlated score large than the threshold *thresh* will be indexed as the true similar document paris in the collection.

The data structure used for the accumulator is very crutial to the overall performance of the score accumulation process. The similarity pair accumlator has far more items than the relevance score accumlator used in IR. In most cases, it could not be held in the main memory of a typical enterprise search server. So the accumlator should be based on second storage. Also, it should support fast insert and lookup even with very large number of items stored.

The current file based map structures that we have in iZENElib such as SDB and HDB etc. all have a characteristic of becoming slow either in insert or in lookup time when it has large number of items stored, and thus could not be used as the score accumlators on such huge data at all. Fortunately, we have a very fast external sort algorithm in iZENElib that is based on OzSort(33) which could be used in some way as the accumulator. Specifically, all the occureneces of candidate pairs for all terms and properties are added into the external sorter. Then, all the items are sorted according to some order on document pair. Finally, on one scan of the sorted result, the scores of the same document pairs on different terms and properties are accumulated. At the same time, as we scanning the result, for each document, its assocaited true similar documents are identified and stored in a file based index to support online query.

4.3.4 Field based Index Pruning

Even using the fast term-at-time score accumulation based on the inverted index, the similarity index problem is still beyond computable when the document number becomes very large. Index pruning is an effective way to reduce the index size and speed up the computation.

We prune the inverted index of the collection from two levels. First, the insignificant terms are pruned from the term lexicon. Second, the insignificant document pairs are pruned from the candidate document pair set with respect to the term posting. Specifically, when using pruning, the candidate pair set becomes:

$$Cand_{t,p} = \begin{cases} \{(d_i, d_j) | d_i, d_j \in posting_{t,p}; d_i \neq d_j; w_{d_i,p}(t) * w_{d_j,p} > \beta\} & \text{if } DF(t) < \alpha \\ \emptyset \text{ otherwise} \end{cases} \tag{4.8}$$

Note that our pruning on the inverted index is property specific. This is one of the key for our multi-property approach for document similarity computation. First, research in IR has observed that the term distribution is very different between different document properties like "title" and "content". For example, while many functional words commonly appear across all document "content", some of which may occur seldomly or unevenly on document titles. So, salient terms with respect to one property are not necessarily important in another property. Second, the lexicon size of different properties vary at a large extent. For example, the lexicon size of one property maybe one hundred times large than another. Also, different properties have different weight. The property specific pruning enables us to prune far more terms on those lengthy but unimportant properties while retain much information as possible on those short but important properties. Thus, it enables us to make a good effectiveness and efficiency trade-off in the document similarity computation problem.

Chapter 5

Query Reminding

5.1 Background

Queries against search engines are natual language expressions of people's information need. It is formed according to the user's knowledge about the interested information focus. However, in this era, new events, problems, challenges and technologies are emerging all the times. Oftentimes, users open a search engine and want to know or learn something while they have limited knowledge or forget how to exactly express their information need.

In recent years, query logs has been widely exploited to improve search engines usability. The underlying intuition is that many people has similar information need. The acumulated queries issued by different people could be viewed as an collaborative knowledge source to refine a current user's query formation towards a specific information focus. For example, by mining the query log, query suggestion features such as query recommendataion, query auto-completion and query correction help users form more accurate queries to better express their information need, and thus obtain a more useful search result. However, most query suggestion technologies applied in the information retrieval society are query specific which suggest related queries based on the user's initial input.

Besides providing the traditional query specific suggestion functions, SF1-R provides the query independent suggestion features which have not been deeply investigated in the IR literature. In developing those features, SF1-R assumes that a search engine also plays the role of a proactive information reminder along

with the role of a passive information provider. For example, when a user opens a search engine, we may remind him as what is the hottest or newest information forcus point for the collectin to be searched. We term such a query suggestion mechanism as query reminding.

Specifically, SF1-R provides two query reminding mechanisms, one is popular query and the other is real-time query. Popular query enable the users to overview the hottest information focus for a given collection while real-time query reminds the user what new focus, problem or event are happening instantly in the user's community. We could illustrate some typical application scenarioes for those query reminding features.

- In the Web search scenario, when a user opens the search engine, he is reminded with the hottest focusing aspects about some news or events just occurred. For example, he maybe reminded by "location of Haiti" in the morning when the breaking "Haiti earthquake" is happened in last night.

- In an enterprise search scenario, queries are usually issued against specific collections. In such cases, those query reminding mechanisms are more useful in locating some new or hot information towards some aspects of the collection. For example, when some events happen in the enterprise, many people may have the same information need towards some specific known item such as a file or report while a person may not know how to search against the file if he is not very familiar with it. In the morning when the person comes to work, he opens the enterprise search engine and possibly be reminded by the file's name or some other information because it is the current interest focus in the user's community. This will save the person a lot of time in finding the information.

- In an enterprise, a person may leave the orgnization for a period because of business or holiday trip or some other personal reasons. When he comes back to work, he can get a quick view on the set of popular queries these days and have a good understanding on the current hot focus in the company.

5.2 Related Works [To be added]

5.3 Algorithm in SF1-R

5.3.1 Notation

A time series data could be defined as $T = (o_1, t_1), (o_2, t_2), ..., (o_m, t_m)$ which is an ordered set of observation o_i with the corresponding time stamps t_i. Query log accumulated day after day could be viewed as a kind of time series data on specified slice windows.

Queries reflect users' information need. During some specific time period, there will typically be some very hot topics or themes which are the most popular information focus points across the user's community. When a new time slice is coming, some novel topics or themes may appear which differentiates them with the past interest. As time goes by, the real time focus point changes and the popular informaton over some specified time period evolves. By query reminding, in SF1-R, we want to reflect the popular topics over some time period as well as the current topic focus in the orgnization through data mining on the query log data.

To better illustrate our algorithms in modeling popular and real-time queries, we give the following definitions with respect to query log.

Definition 5.3.1. *Slice Query Log: Given a time slice t_i, a slice query log L_i with respect to t_i is $\{(q_j, f(q_j))|1 \leq j \leq n, q_j \in QueryAt(t_i)\}$, where $QueryAt(t_i)$ is the set of queries occurred in the time slice t_i; $q_j = w_{j_1}, w_{j_2}, ..., w_{j_l}$ is a query formed by l words; $f(q_j)$ is the overall occurring frequency of q_j in the time slice. Slice query log is the smallest time unit we model in the query log time series.*

Definition 5.3.2. *Period Query Log: Given a time period constituted by a continuous sequence of time slices $t = t_1, t_2, ..., t_m$, a period query log L with respect to t is defined as $L = L_1, L_2, ..., L_m$.*

5.3.2 Measuring Popularity

Given a period query log L over $t = t_1, t_2, ..., t_m$, the task of popular query reminding is to identify a ranked list of queries from L that characterize the most

popular information focus during the time period.

For a query q_i, the relative frequency $f(q_i)$ in L is the most intuitive and direct way to model the popularity of q_i in L. However, it is not very effective to use the relative frequency alone. First, in an enterprise search environment, the query log data is relatively sparse. Second, a similar information need could be expressed in numerous different ways. Finally, as well known, using relative frequency alone will typically result in many common and short queries.

To better model a query's popularity in a time period, besides the relative frequency, we use the generation probability of the language model estimated from the period query log as another important ranking criteria. A language model is a probabilistic distribution $\{p(w|\theta_i)|w \in V\}$ over words in a vocabulary V. Exploiting the idea of multiple prior weighting in language modeling proposed by PLM (34), we use a variation entropy as an important bias factor to weight the estimation of the term's emission probability.

For the period query log $L = L_1, L_2, ..., L_m$, the word w's variation entropy is defined as:

$$VE(w) = -\sum_{i=1}^{m} \frac{f_i(w)}{F_w} log \frac{f_i(w)}{F_w} \tag{5.1}$$

where the index i represents the ID of different time slices in the time period; $f_i(w)$ is the overall occurrence frequency of w in the slice query log L_i associated with t_i; $F_w = \sum_{i=1}^{m} f_i(w)$ means the overall frequency of term w in the period query log.

The word's variation entropy is used as a popularity weighting parameter on the word's emission probability. For two words w_1 and w_2, given that other statistics associated with them being equal (such as having identitcal appearance frequency in the time period) while w_1 appears more consistently than w_2 in those different time slices of a time period, then w_1 should be more popular than w_2 with respect to the time period.

Given such prior weight, after simplification on this specific case according to the PLM ranking method, the generation probability of a query $q = (w_1, w_2, ..., w_l)$ could be written as

$$PopGen(q) = \sum_{1 \le j \le l} log \frac{F(w_j) + \lambda VE(w_j) + \mu p(w_j|C)}{|L| + \sum_{i=1}^{|V|} \lambda VE(w_i) + \mu} \tag{5.2}$$

where $|L|$ is the total number of word occurrences in L; $p(w_j|C)$ is an background language model used to discount some common words.

Then, the overall popularity for the query q with respect to L is modeled as a combination of the generation probability and the relative frequency of q in L.

$$Popularity(q) = PopGen(q) * f(q) \tag{5.3}$$

5.3.3 Measuring Novelty

While popular queries are measured with respect to a period query log, we model real-time queries on the latest slice query log. To reflect the new appearing information focuses, on one hand, real time queries reminded should be hot topics locally in the slice query log. On the other hand, more importantly, it should be novel in some extent. The real-time queries will be reminded for the user frequently after each new query log slice is formed, novelty should be a very important aspect in measuring real-time queries. Users will surely get bored with the continuously reminded similar content.

Specifically, we use some period query log in the past before the latest slice query log as referential points to measure a query's novelty. Let l_i be the latest slice query log, a referential period query log with respect to l_i is $L_r = l_{i-m}, ..., l_{i-2}, l_{i-1}$. Then, for a query $q = (w_1, w_2, ..., w_l)$, it's novelty is measured by the following divergence score.

$$Novelty(q) = \sum_{1 \leq j \leq l} f_i(w_j) log \frac{f_i(w_j)}{F(w_j)} \tag{5.4}$$

where $f_i(w_j)$ is the frequency of word w_j appeared in the slice query log l_i while $F(w_j)$ is the frequency of w_j in the referential period query log.

Then, the ranking of real-time queries is measured as

$$Rank(q) = Novelty(q) * f_i(q) \tag{5.5}$$

where $f_i(q)$ is the relative frequency of q occurred in the slice query log l_i.

Chapter 6

Additional Research

6.1 Topic Modeling and Its Application

6.1.1 Background

Topic models are Bayesian models for a collection of documents that explain the observed collection with a small set of distributions over terms. Those distributions tend to correspond to the intuitive notions of the themes or topics underlying the documents. Topic models have been approved to be a very powerful tool for unsupervised analysis of text and have been successfully applied in variety of fields.

The idea underlying topic models is to suppose a probabilistic process in which both a hidden topic structure and observed document data arises. Given the observed documents, one could then inference this process to determine the posterior distribution of the hidden topic structure. Topic models are motivated and built on techniques such as latent semantic analysis (LSA) (35) and probabilistic latent semantic analysis (pLSA) (36). Topic models improves over LSA and PLSA by adopting a fully generative model and attain better generalization and are easily extendible.

6.1.1.1 Latent Dirichlet Allocation

We first introduce Latent Dirichlet Allocation (LDA) (37) which is the most commonly used building block for topic models.

LDA represents documents as random mixtures over latent topics, where each topic is defined to be a distribution over a a fixed vocabulary of terms. Specifically, LDA assumes that Z topics are associated with a collection, and each document exhibits those topics with different proportions. This is a natural assumption because each document may combine a different subset of themes with respect to the whole collection.

LDA is a hidden variable model where the observed data are the words of each document and the hidden variables are the latent topical structure, i.e., the topics themselves and how each document exhibits them. The statistical assumptions underlying LDA could be understood by its probabilistic generative process. Notations in the following description could be referred to Table **??**.

1. Pick a multinomial distribution $\vec{\phi}_z$ for each topic z from a Dirichlet distribution with parameter $\vec{\beta}$;

2. For each document d, pick a multinomial distribution $\vec{\theta}_d$ from a Dirichlet distribution with parameter $\vec{\alpha}$.

3. For each word token w_i in document d, first pick a topic $z_{d,i} \in 1...Z$ from the multinomial distribution $\vec{\theta}_d$;

4. and then pick the word $w_{d,i}$ from the multinomial distribution $\vec{\phi}_z$.

Given the collection data, the posterior distribution of the hidden variables given the words of the documents provides a probabilistic decomposition of the documents

$$p(\Phi, \Theta, Z|W, \vec{\alpha}, \vec{\beta}) \tag{6.1}$$

However, the exact inference of the posterior probability is intractable. The solution to this is to use approximate inference algorithms, such as mean-field variational maximisation (38), expectaion propagation (39) and Gibbs sampling (40; 41).

In recent years, many extentions over LDA have been done based on the basic features of LDA. We review some of the works in several most related directions with our research field in the following.

6.1.1.2 Adding correlation between topics

In LDA, the topic proportions for specific document are draw from a single non-informative Dirichlet prior. A limitation of LDA is the inability to model topic correlation. For example, a document about sports is more likely to also be about health han international finance. So, a natural extending direction for LDA is to introduce correlation between topics.

(42) follows the non-parametric Bayesian approach and proposes the hierachical Dirichlet process model (HDP). This model uses the hierarchical Dirichlet process as prior instead of a single Dirchlet distribution on the topic mixture. It assumes that there are groups of data, and that the infinite mixture components are shared among these groups. Thus, it allows dependencies across groups to be modeled effectively as well as conferring generalization to new groups.

(43) develops the correlated topic model (CTM), where the topic proportions exhibit correlation via the logistic norm distribution. However, the inference problem of this model is complicated because of the fact that the logistic normal is not conjugate to the multinomial.

(44) proposes the HLDA model that represents the distribution of topics within documents by organizing the topics into a tree. Then, documents are assumed to be generated by the topics along a single path of the topic tree. The structure of the tree is learned along with the topics themselves using a nested Chinese restaurant process.

(45) introduces the Pachiko allocation model (PAM), which uses a directed acyclic graph structure to represent and learn arbitrary-arity topic correlations. In PAM, the concept of topics are extended to be distributions not only over words, but also over other topics. PAM therefore captures the correlation among topics. However, PAM does not represent a nested hierachy of topics. (46) presents hierachical PAM (HPAM) that explicitly represents a topic hierachy. This model can be seen as comining the advantages of hLDA's topical hierachy representation with PAM.

6.1.1.3 Considering Word Order

As many other statistical models in text processing, LDA adapts the bag of words assumption by assuming that words are generated independently from each other. Some works are done to modify LDA modification under the requirement for some specific tasks in which the word order information is important.

Some works try to include bigram, or phrases into topic modeling. (47) proposes bigram topic model (BTM) by including properties of a hierachical Dirichlet bigram language model to LDA. In this model, the generation of a word for a topic always depends on the previous word. (48) proposes LDA collocation model (LDACOL) which introduces a new set of variables that denote if a bigram can be formed with the previous token. Thus, it attains the power to determine whether to generate a bigram or a unigram. However, in LDACOL collocations are thought to be same across different topics. (49) generalizes BTM and LDACOL to topical N-gram model (TNG) that considers the topic dependence for forming meaningful phrases. For example, this model can view "white house" as a special meaning phrase in the "politics" topic, but not in the "real state" topic.

Another stream of effort tries to modeling word order by incorprating the commonly used sequential statistical models into topic modeling. (50) incorporates a Hidden Markov Model (HMM) to model short range syntactic constrints between words. As a consequence, HMM-LDA is able to simultaneously learn syntactic classes and semantic topics and identify the role that words play in documents. (51) proposes hidden topic markov models (HTM) to model the topics in the document as Markov chain. Specifically, it assumes the words from the same sentence have the same topic, and successive sentences are more likely to have the same topics.

6.1.1.4 Adding Supervised Mechanisms

LDA is a totally unsupervised model which exploits only the word information of adocuments to derive structured topics. Recently, some attempts have been made to add some supervised information to LDA.

(52) introduces supervised latent Dirichlet allocation (sLDA) which exploits the labels of document in topic modeling. The model accommodates a variety

of response types. Specifically, they assume that the words and their topic assignments of a document is generated first. Then, based on the document, the response variable is generated. The model is tested on real-world problems such as movie ratings predicted from reviews, and sLDA shows better performance than the unsupervised LDA followed by a separate regression.

(53) proposes a Dirichlet-multinomial regression model (DMR) which can handle various types of side information, including the case in which this side information is an indicator variable of the class. Different from sLDA, DMR is fully conditional with respect to the observed features of the document.

(54) introduces the multi-grain LDA (MG-LDA) in which two sets of topics are defined, one is the common global topic in LDA, the other is the local topics asssociates with aspects. This model is used in aspect ratings for sentiment summarization. The supervision available from aspect rating has been added to the model as observed variable which affects the final topic assignment in the model.

(55) presents a discriminative framework (DiscLDA) to exploit the supervised side information in topic modeling. In this model, a class-dependent linear transformation is introduced on the topic mixture proportions. The transformed topic mixture proportions are used as a new representation of documents which preserves the predictive power for the task of classification.

6.1.1.5 Modeling Document MetaData

Besides the document labels, other side document information has been widely exploited to improve over LDA.

(56) proposes the author topic model. In this model, the author information of the document is viewed as an important factor in generating text. Words are generated by first selecting an author uniformly from an observed author list and then selecting a topic from a distribution over topics that is specific to that author. Given a topic, words are selected as before. This model assumes that each word is generated by one and only one author.

(57) introduces the topics over time model which incorporates the document's time information into topic modeling. By this, the model captures not only the

low-dimensional structure of data, but also hwo the structure changes over time. Specifically, each topic is associated with a continuous distribution over timestamps, and for each generated document, the mixture distribution over topics is influenced by both word co-occurrences and the document's timestamp.

(58) presents the entity topic model which aims to model the named entities along with other words in the document text. Named entities are generated automatically in a set of articles. They use the learned probability distributions to do the predictions and evaluate by the average best rank.

(59) proposes the mixture membership model that incorporates reference link informaiton into topic modeling. In such model, a similar generative process of hyperlinks is added in parallel with the generation of words. The same document specific topic distribution is used to generate both words and hyperlinks. Thus, this model captures the notion that documents sharing the same hyperlinks and same words, tend to be on the same topic.

6.1.2 Positional LDAs

6.1.2.1 Model Description

In this model, we consider the simplest case that we think the meta position information of the word occurrence has some influence on its topic assignment. The meta position information of words could include information like whether the word is occurred in the beginning or ending of a sentence. Whether it is occurred in the first sentence of a paragraph, etc. We assume a fixed vocabulary of descriptors O of a word's possible meta position occurrence information.

The following generative process is assumed in pLDA1.

1. Draw a multinomial distribution $\vec{\phi}_z$ for each topic z from a Dirichlet distribution with parameter $\vec{\beta}$; Draw a multinomial distribution $\vec{\psi}_z$ for each topic z from a Dirichlet distribution with parameter $\vec{\gamma}$;

2. For each document d, draw a multinomial distribution $\vec{\theta}_d$ from a Dirichlet distribution with parameter $\vec{\alpha}$.

3. For each word token w_i in document d, pick a topic $z_{d,i} \in 1...Z$ from the multinomial distribution $\vec{\theta}_d$;

4. Pick the position descriptor $o_{d,i}$ from the multinomial distribution $\vec{\psi}_{z_{d,i}}$.

5. Pick the word $w_{d,i}$ from the multinomial distribution $\vec{\phi}_{z_{d,i}}$.

6.1.2.2 Parameter Estimation

Like LDA, the exact inference is intractable for pLDA1, we use Gibbs sampling for the approximate inference. Gibbs sampling is a special case of Markov-chain Monte Carlo (MCMC). MCMC methods could emulate high-dimensional probability distribution $p(\vec{x})$ by the stationary behaviour of a Markov chain. This means that one sample is generated for each transition in the chain after a stationary state of the chain has been reached, which happens after a so-called "burn-in" period that eliminates the influence of initialization. Gibbs sampling is a special case of MCMC where the dimension x_i of the distribution are sampled alternately on at a time, conditioned on the values of all other dimensions $\vec{x}_{-i}$.

For models that contain hidden variables $\vec{z}$ like LDA, their posterior given the evidence, $p(\vec{z}|\vec{x})$, is a distribution commonly wanted. The full conditionals must be found to build a Gibbs sampler. The general formulation of a Gibbs sampler for such latent-variable models is:

$$p(z_i|\vec{z}_{-i},\vec{x}) = \frac{p(\vec{z},\vec{x})}{p(\vec{z}_{-i},\vec{x})} = \frac{p(\vec{z},\vec{x})}{\int_Z p(\vec{z},\vec{x})dZ_i} \tag{6.2}$$

In pLDA1, there are two sets of observed variables W and O and one set of latent variables Z. Following the above general procedure, we build the Gibbs sampler for pLDA1 by first modeling the joint distribution $p(Z,W,O|\alpha,\beta,\gamma)$

$$\begin{aligned}
&p(Z,W,O|\vec{\alpha},\vec{\beta},\vec{\gamma}) \\
&= p(Z|\vec{\alpha})p(W|Z,\vec{\beta})p(O|Z,\vec{\gamma}) \\
&= \int p(Z|\Theta)p(\Theta|\vec{\alpha})d\Theta \cdot \int p(W|\Phi,Z)p(\Phi|\vec{\beta})d\Phi \cdot \int p(W|\Psi,Z)p(\Psi|\vec{\gamma})d\Psi \\
&= \int \prod_{d=1}^{D} \tfrac{1}{\triangle(\vec{\alpha})} \prod_{z=1}^{Z} \theta_{d,z}^{n_d^{(z)}+\alpha_z-1} d\vec{\theta}_d \cdot \int \prod_{z=1}^{Z} \tfrac{1}{\triangle(\vec{\beta})} \prod_{i=1}^{V} \phi_{z,i}^{n_z^{(i)}+\beta_i-1} d\vec{\phi}_z \cdot \\
&\quad \int \prod_{z=1}^{Z} \tfrac{1}{\triangle(\vec{\gamma})} \prod_{j=1}^{O} \psi_{z,j}^{m_z^{(j)}+\gamma_j-1} d\vec{\psi}_z \\
&= \prod_{d=1}^{D} \tfrac{\triangle(\vec{n}_d+\vec{\alpha})}{\triangle(\vec{\alpha})} \cdot \prod_{z=1}^{Z} \tfrac{\triangle(\vec{n}_z+\vec{\beta})}{\triangle(\vec{\beta})} \cdot \prod_{z=1}^{Z} \tfrac{\triangle(\vec{m}_z+\vec{\gamma})}{\triangle(\vec{\gamma})}
\end{aligned} \tag{6.3}$$

where

$$\triangle(\vec{x}) = \frac{\prod_{i=1}^{K}\Gamma(x_i)}{\Gamma(\sum_{i=1}^{K}x_i)} \tag{6.4}$$

Given the joint distribution, we can get the full conditional distribution for a word token.

$$\begin{aligned} p(z_i|Z_{-i},W,O) &= \frac{p(Z,W,O)}{p(Z_{-i},W,O)} = \frac{p(Z)}{p(Z_{-i})} \cdot \frac{p(W|Z)}{p(W_{-i}|Z_{-i})p(w_i)} \cdot \frac{p(O|Z)}{p(O_{-i}|Z_{-i})p(o_i)} \\ &\propto \frac{\triangle(\vec{n}_d+\vec{\alpha})}{\triangle(\vec{n}_{d_{-i}}+\vec{\alpha})} \cdot \frac{\triangle(\vec{n}_z+\vec{\alpha})}{\triangle(\vec{n}_{z_{-i}}+\vec{\alpha})} \cdot \frac{\triangle(\vec{m}_z+\vec{\alpha})}{\triangle(\vec{m}_{z_{-i}}+\vec{\alpha})} \\ &\propto \frac{n_d^{(z)}+\alpha_z-1}{\sum_{z=1}^{Z}(n_d^{(z)}+\alpha_z)-1} \cdot \frac{n_z^{(i)}+\beta_i-1}{\sum_{i=1}^{V}(n_z^{(i)}+\beta_i)-1} \cdot \frac{m_z^{(j)}+\gamma_j-1}{\sum_{j=1}^{V}(m_z^{(j)}+\gamma_j)-1} \end{aligned} \tag{6.5}$$

with a set of samples from the posterior distribution $p(Z|W,O)$, we could get the predictive probability of the set of model parameters.

$$\hat{\theta}_d^z = \frac{n_d^{(z)} + \alpha_z}{\sum_{z=1}^{Z}(n_d^{(z)} + \alpha_z)} \tag{6.6}$$

$$\hat{\phi}_z^i = \frac{n_z^{(i)} + \beta_i}{\sum_{i=1}^{V}(n_z^{(i)} + \beta_i)} \tag{6.7}$$

$$\hat{\psi}_z^j = \frac{m_z^{(j)} + \gamma_j}{\sum_{j=1}^{V}(m_z^{(j)} + \gamma_j)} \tag{6.8}$$

6.1.3 A Bayesian Model for Compound Extraction

6.1.3.1 Model Description

In this section, we apply the bayesian method commonly used in topic modeling to the problem of extraction of compounds from the text collections. A compound is constituted by several simple words which joined sequentially to denote a new concept that is more specific than its constituent words. The labels we used in taxonomy generation of SF1 are all compounds which plays a key role in the data mining components of SF1. The consituents in a compound could typically be differentiated by modifiers and head. Typically, the right most word in a compound is the head and the words before head are modifiers.

There are two basic assumptions in compound formation. First, different words has different ability to compound with other words. This property is called compounding ability of words. Second, different words has their own preference to occur as modifier or head in certain contexts. Based on those two basic assumption, we derive a Bayesian compound extraction (BCE) model.

The basic idea of BCE is that we assume each word has an associated latent label which indicates the different roles it plays in the text with respect to compound formation. Specifically, three different categories are differentiatied, "out", "modifer" and "head" (denoted by c_1, c_2 and c_3 respectively). "out" means the word is out of any compound; "modifier" and "head" means the word is in the modifier and head position of a compound respectively. Then, each word is assumed to be generated according to its category and a local context which is expressed by the former word in the current model.

The detailed generative process of the model is illustrated as follows.

1. For category $c_j (j = 1, 2, 3)$ and each word w_i, draw multinomial distribution $\vec{\phi}_{c_j,w_i}$ from the respective Dirichlet prior $\vec{\beta}_{c_j}$.

2. For each word w_i in the collection, draw multinomial distribution $\vec{\psi}_{w_i}$ on vocabulary C from a Dirichlet prior $\vec{\gamma}$.

3. For each word w_i in the collection:

   3.1 draw $x_i \sim multinomial(\vec{\psi}_{w_{i-1}})$.

   3.2 if $x_i = c_1$, draw w_i from $\vec{\phi}_{c_1,w_{i-1}}$;
   if $x_i = c_2$, draw w_i from $\vec{\phi}_{c_2,w_{i-1}}$;
   if $x_i = c_3$, draw w_i from $\vec{\phi}_{c_3,w_{i-1}}$;

6.1.3.2 Parameter Estimation

There are four sets of latent variables in the model, i.e. Φ_{c_1}, Φ_{c_2} , Φ_{c_3} and Ψ. We could express the joint probability as:

$$
\begin{aligned}
&p(X, W|\vec{\beta}_{c_1}, \vec{\beta}_{c_2}, \vec{\beta}_{c_3}, \vec{\gamma}) \\
&= p(X|\vec{\gamma}) \cdot p(W|X, \vec{\beta}_{c_1}, \vec{\beta}_{c_2}, \vec{\beta}_{c_3}) \\
&= \iiiint_{\Psi,\Phi_1,\Phi_2,\Phi_3} p(X|\Psi, W)p(\Psi|\vec{\gamma})p(\Phi_1|\vec{\beta}_{c_1})p(\Phi_2|\vec{\beta}_{c_2})p(\Phi_3|\vec{\beta}_{c_3}) \\
&p(W|X, \Phi_1, \Phi_2, \Phi_3)d\Psi d\Phi_1 d\Phi_2 d\Phi_3 \\
&= \int_\Psi \prod_{i=1}^{W} \prod_{k=1}^{3} \psi_{ik}^{N_{ik}+\gamma_k-1} d\psi \cdot \int_{\Phi_1} \prod_{i=1}^{W} \prod_{j=1}^{W} \phi_{1,ij}^{B_{ij}+\beta_{c_1,j}-1} d\Phi_1 \cdot \\
&\int_{\Phi_2} \prod_{i=1}^{W} \prod_{j=1}^{W} \phi_{2,ij}^{M_{ij}+\beta_{c_2,j}-1} d\Phi_2 \cdot \int_{\Phi_3} \prod_{i=1}^{W} \prod_{j=1}^{W} \phi_{3,ij}^{B_{ij}+\beta_{c_3,j}-1} d\Phi_3 \\
&= \prod_i \triangle(\vec{N}_i + \vec{\gamma}) \cdot \prod_i \triangle(\vec{B}_i + \vec{\beta}_{c_1}) \cdot \prod_i \triangle(\vec{M}_i + \vec{\beta}_{c_2}) \cdot \prod_i \triangle(\vec{H}_i + \vec{\beta}_{c_3})
\end{aligned}
\tag{6.9}
$$

Given the joint distribution, we can get the full conditional distribution for a word token.

$$
\begin{aligned}
p(x_i|X_-,W) &= \frac{p(X,W|\vec{\beta}_{c_1},\vec{\beta}_{c_2},\vec{\beta}_{c_3},\vec{\gamma})}{p(X_-,W|\vec{\beta}_{c_1},\vec{\beta}_{c_2},\vec{\beta}_{c_3},\vec{\gamma})} \\
&\propto \frac{\triangle(\vec{N}_i+\vec{\gamma})}{\triangle(\vec{N}_i^-+\vec{\gamma})} \cdot \frac{\triangle(\vec{B}_i+\vec{\beta}_{c_1})}{\triangle(\vec{B}_i^-+\vec{\beta}_{c_1})} \cdot \frac{\triangle(\vec{M}_i+\vec{\beta}_{c_2})}{\triangle(\vec{M}_i^-+\vec{\beta}_{c_2})} \cdot \frac{\triangle(\vec{H}_i+\vec{\beta}_{c_3})}{\triangle(\vec{H}_i^-+\vec{\beta}_{c_3})} \\
&= \frac{N_{w_{i-1}x_i}+\gamma_{x_i}-1}{\sum_x(N_{w_{i-1}x_i}+\gamma_{x_i})-1} \cdot \begin{cases} \frac{B_{w_{i-1}w_i}+\beta_{c_1,w_i}-1}{\sum_i(B_{w_{i-1}w_i}+\beta_{c_1,w_i})-1} & \text{if } x_i = c_1 \\ \frac{M_{w_{i-1}w_i}+\beta_{c_2,w_i}-1}{\sum_i(M_{w_{i-1}w_i}+\beta_{c_2,w_i})-1} & \text{if } x_i = c_2 \\ \frac{H_{w_{i-1}w_i}+\beta_{c_3,w_i}-1}{\sum_i(H_{w_{i-1}w_i}+\beta_{c_3,w_i})-1} & \text{if } x_i = c_3 \end{cases}
\end{aligned} \tag{6.10}
$$

6.1.4 A Generalized Bayesian Labeling Model

Given a corpus C which contains set of word sequences, we propose the following generative model for it.

1. Generate the category according to the previous category and the previous word.
2. Generate the first word inside that category, conditioning on the current and previous category.
3. Generate the first word feature, conditioning on the current and previous category.
4. Generate all subsequent words inside the current category, where each subsequent word is conditioned on its immediate predecessor.
5. Generate all subsequent word feature according to the current category and the predecessor's word feature.

More formally, we describe its generative process as:

1. For category c_j and each word w_i, draw multinomial distribution $\vec{\phi}_{c_j,w_i}$ on vocabulary C from the Dirichlet prior $\vec{\beta}$.
2. For each category pair c_j and c_k, draw multinomial distribution $\vec{\theta}_{c_j,c_k}$ on vocabulary W from the Dirichlet prior $\vec{\alpha}$.

3. For each category c_j and each word w_i, draw multinomail distribution $\vec{\psi}_{c_j,w_i}$ on vocabulary W from the Dirichlet prior $\vec{\gamma}$.

4. For each category pair c_j and c_k, draw multinomial distribution $\vec{\sigma}_{c_j,c_k}$ on vocabulary F from the Dirichlet prior $\vec{\delta}$.

5. For each category c_j and each feature f_i, draw multinomail distribution $\vec{\lambda}_{c_j,f_i}$ on vocabulary F from the Dirichlet prior $\vec{\eta}$.

6. For each word position w_i in the corpus:

   6.1 Draw $c_i \sim multinomial(\vec{\phi}_{c_{i-1},w_{i-1}})$.

   6.2 If $c_i \neq c_{i-1}$:

      6.2.1 draw w_i from multinomial($\vec{\theta}_{c_{i-1},c_i}$).

      6.2.2 draw f_i from multinomial($\vec{\sigma}_{c_{i-1},c_i}$).

   6.3 If $c_i = c_{i-1}$

      6.2.1 draw w_i from multinomial($\vec{\psi}_{c_i,w_{i-1}}$).

      6.2.2 draw f_i from multinomial($\vec{\lambda}_{c_i,f_{i-1}}$).

6.1.4.1 Parameter Estimation

$$
\begin{aligned}
&p(C,W,F|\vec{\beta},\vec{\alpha},\vec{\delta},\vec{\gamma},\vec{\eta}) \\
&= \iiiint_{\Phi,\Theta,\Sigma,\Psi,\Lambda}[p(\Phi|\vec{\beta})p(C|\Phi,W)][p(\Theta|\vec{\alpha})p(\Psi|\vec{\gamma})p(W|\Theta,\Psi,C)][p(\Sigma|\vec{\delta})p(\Lambda|\vec{\eta})p(F|\Sigma,\Lambda,C)] \\
&d\Phi d\Theta d\Sigma d\Psi d\Lambda \\
&= \int_\Phi \prod_{k=1}^{C}\prod_{i=1}^{W}\prod_{k=1}^{C}\phi_{kil}^{N_{kil}+\beta_l-1}d\Phi \cdot \int_\Theta \prod_{k=1}^{C}\prod_{l=1}^{C}\prod_{i=1}^{W}\theta_{kli}^{U_{kli}+\alpha_i-1}d\Theta \\
&\cdot \int_\Sigma \prod_{k=1}^{C}\prod_{l=1}^{C}\prod_{i=1}^{F}\sigma_{kli}^{V_{kli}+\delta_i-1}d\Sigma \cdot \int_\Psi \prod_{k=1}^{C}\prod_{j=1}^{W}\prod_{i=1}^{W}\psi_{kji}^{X_{kji}+\gamma_i-1}d\Psi \\
&\cdot \int_\Lambda \prod_{k=1}^{C}\prod_{j=1}^{W}\prod_{i=1}^{W}\lambda_{kji}^{Y_{kji}+\eta_i-1}d\Lambda \\
&= \prod_{k,i}\triangle(\vec{N}_{ki}+\vec{\beta}) \cdot \prod_{k,l}\triangle(\vec{U}_{kl}+\vec{\alpha}) \cdot \prod_{k,l}\triangle(\vec{V}_{kl}+\vec{\delta}) \cdot \prod_{k,i}\triangle(\vec{X}_{ki}+\vec{\gamma}) \cdot \prod_{k,i}\triangle(\vec{Y}_{ki}+\vec{\eta})
\end{aligned}
\tag{6.11}
$$

Given the joint distribution, we can get the full conditional distribution for a

word token.

$$
\begin{aligned}
&p(c_i|C_-, W, F) = \frac{p(C,W,F|\vec{\beta},\vec{\alpha},\vec{\delta},\vec{\gamma},\vec{\eta})}{p(C_-,W,F|\vec{\beta},\vec{\alpha},\vec{\delta},\vec{\gamma},\vec{\eta})} \\
&\propto \frac{\triangle(\vec{N}_{ki}+\vec{\beta})}{\triangle(\vec{N}_{ki}^{-}+\vec{\beta})} \cdot \frac{\triangle(\vec{U}_{kl}+\vec{\alpha})}{\triangle(\vec{U}_{kl}^{-}+\vec{\alpha})} \cdot \frac{\triangle(\vec{V}_{kl}+\vec{\delta})}{\triangle(\vec{V}_{kl}^{-}+\vec{\delta})} \cdot \frac{\triangle(\vec{X}_{ki}+\vec{\gamma})}{\triangle(\vec{X}_{ki}^{-}+\vec{\gamma})} \cdot \frac{\triangle(\vec{Y}_{ki}+\vec{\eta})}{\triangle(\vec{Y}_{ki}^{-}+\vec{\eta})} \\
&= \frac{N_{c_{i-1}w_{i-1}c_i}+\beta_{c_i}-1}{\sum_c(N_{c_{i-1}w_{i-1}c_i}+\beta_{c_i})-1} \cdot \begin{cases} \frac{U_{c_{i-1}c_iw_i}+\alpha_{w_i}-1}{\sum_w(U_{c_{i-1}c_iw_i}+\alpha_{w_i})-1} \cdot \frac{V_{c_{i-1}c_if_i}+\delta_{f_i}-1}{\sum_f(V_{c_{i-1}c_if_i}+\delta_{f_i}-1)-1} & \text{if } c_i \neq c_{i-1} \\ \frac{X_{c_iw_{i-1}w_i}+\gamma_{w_i}-1}{\sum_w(X_{c_iw_{i-1}w_i}+\gamma_{w_i}-1)-1} \cdot \frac{Y_{c_if_{i-1}f_i}+\eta_{f_i}-1}{\sum_f(Y_{c_if_{i-1}f_i}+\eta_{f_i})-1} & \text{if } c_i = c_{i-1} \end{cases}
\end{aligned}
\tag{6.12}
$$

in which, W and F represents the word occurrence vector and feature occurrence vector of the corpus respectively. $N_{c_{i-1}w_{i-1}c_i}$ means number of times c_i appears as the next category after the previous category c_{i-1} and previous word w_{i-1}; $U_{c_{i-1}c_iw_i}$ means number of times w_i occurs as the first word in current category c_i in the case of the previous category is c_{i-1}; $V_{c_{i-1}c_if_i}$ means number of times feature f_i occurs as the first word feature in current category c_i in the case of the previous category is c_{i-1}; $X_{c_iw_{i-1}w_i}$ means number of times word w_i occurs after word w_{i-1} in the current category c_i; $Y_{c_if_{i-1}f_i}$ means number of times word feature f_i occurs after word feature f_{i-1} in the current category c_i.

6.1.4.2 Selecting Word Features

Current, we attempt the following most frequently used word features in NER.

- Number: the word is a number
- All upper-case: all letters are upper case.
- All lower-case: all letters are lower case.
- First lower-case: first letter are lower case.
- First sentence word: the word is at the begining of a sentence.

References

[1] Frank, E., Paynter, G., Witten, I., Gutwin, C., Nevill-Manning, C.: Domain-specific keyphrase extraction. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE. Volume 16., LAWRENCE ERLBAUM ASSOCIATES LTD (1999) 668–673

[2] Turney, P.: Learning algorithms for keyphrase extraction. Information Retrieval **2**(4) (2000) 303–336

[3] Steier, A., Belew, R.: Exporting phrases: A statistical analysis of topical language. In: Second Symposium on Document Analysis and Information Retrieval, Citeseer (1993) 179–190

[4] Tomokiyo, T., Hurst, M.: A language model approach to keyphrase extraction. In: Proceedings of the ACL Workshop on Multiword Expressions. (2003) 33–40

[5] Chien, L.: PAT-Tree-Based Adaptive Keyphrase Extraction for Intelligent Chinese Information Retrieval. Information Processing & Management **35**(4) (1999) 501–21

[6] Zamir, O., Etzioni, O.: Web document clustering: A feasibility demonstration. In: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, ACM New York, NY, USA (1998) 46–54

[7] Ferragina, P., Gulli, A.: A personalized search engine based on web-snippet hierarchical clustering. In: International World Wide Web Conference, ACM New York, NY, USA (2005) 801–810

[8] Hearst, M., Pedersen, J.: Reexamining the cluster hypothesis: scatter/gather on retrieval results. In: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval, ACM Press New York, NY, USA (1996) 76–84

[9] Giannotti, F., Nanni, M., Pedreschi, D., et al.: Webcat: Automatic categorization of web search results. In: Proceedings of 11th Italian Symposium on Advanced Database Systems. 507–518

[10] Geraci, F., Pellegrini, M., Maggini, M., Sebastiani, F.: Cluster generation and cluster labelling for web snippets

[11] Mei, Q., Shen, X., Zhai, C.: Automatic labeling of multinomial topic models. In: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM New York, NY, USA (2007) 490–499

[12] Zamir, O., Etzioni, O.: Grouper: a dynamic clustering interface to Web search results. Computer Networks-the International Journal of Computer and Telecommunications Networkin **31**(11) (1999) 1361–1374

[13] Osiriski, S., Weiss, D.: Conceptual clustering using lingo algorithm: Evaluation on open directory project data. In: Intelligent Information Processing And Web Mining: Proceedings of the International IIS: IIPWM'04 Conference Held in Zakopane, Poland, May 17-20, 2004, Springer (2004) 369

[14] Zeng, H., He, Q., Chen, Z., Ma, W., Ma, J.: Learning to cluster web search results. In: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval, ACM New York, NY, USA (2004) 210–217

[15] Maarek, Y., Fagin, R., Ben-Shaul, I., Pelleg, D.: Ephemeral document clustering for web applications. IBM research report RJ **10186** (2000)

[16] Lawrie, D., Croft, W.: Generating hierarchical summaries for web searches. In: Proceedings of the 26th annual international ACM SIGIR conference

on Research and development in informaion retrieval, ACM New York, NY, USA (2003) 457–458

[17] Kummamuru, K., Lotlikar, R., Roy, S., Singal, K., Krishnapuram, R.: A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In: Proceedings of the 13th international conference on World Wide Web, ACM New York, NY, USA (2004) 658–665

[18] Kummamuru, K., Lotlikar, R., Roy, S., Singal, K., Krishnapuram, R.: A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In: WWW '04: Proceedings of the 13th international conference on World Wide Web, New York, NY, USA, ACM (2004) 658–665

[19] Salton, G., Wong, A., Yang, C.: A vector space model for automatic indexing. Communications of the ACM **18**(11) (1975) 620

[20] Lafferty, J., Zhai, C.: Document language models, query models, and risk minimization for information retrieval. In: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, ACM (2001) 111–119

[21] Elsayed, T., Lin, J., Oard, D.: Pairwise document similarity in large collections with MapReduce. In: Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers, Association for Computational Linguistics (2008) 265–268

[22] Broder, A., Glassman, S., Manasse, M., Zweig, G.: Syntactic clustering of the web. Computer Networks and ISDN Systems **29**(8-13) (1997) 1157–1166

[23] Fetterly, D., Manasse, M., Najork, M.: On the evolution of clusters of near-duplicate web pages. In: Web Congress, 2003. Proceedings. First Latin American. (2003) 37–45

[24] Charikar, M.: Similarity estimation techniques from rounding algorithms. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, ACM (2002) 388

[25] Henzinger, M.: Finding near-duplicate web pages: a large-scale evaluation of algorithms. In: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, ACM (2006) 284–291

[26] Turtle, H., Flood, J.: Query evaluation: strategies and optimizations. Information Processing & Management **31**(6) (1995) 831–850

[27] Buckley, C., Lewit, A.: Optimization of inverted vector searches. In: Proceedings of the 8th annual international ACM SIGIR conference on Research and development in information retrieval, ACM (1985) 110

[28] Robertson, S., Zaragoza, H., Taylor, M.: Simple BM25 extension to multiple weighted fields. In: Proceedings of the thirteenth ACM international conference on Information and knowledge management, ACM (2004) 42–49

[29] Ogilvie, P., Callan, J.: Combining document representations for known-item search. In: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, ACM (2003) 150

[30] Croft, W., Turtle, H., Lewis, D.: The use of phrases and structured queries in information retrieval. In: Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval, ACM (1991) 32–45

[31] Fagan, J.: Automatic phrase indexing for document retrieval. In: Proceedings of the 10th annual international ACM SIGIR conference on Research and development in information retrieval, ACM (1987) 91–101

[32] Robertson, S., Walker, S., Jones, S., Hancock-Beaulieu, M., Gatford, M.: Okapi at TREC-4. In: Proceedings of the Fourth Text Retrieval Conference. (1996) 73–97

[33] Sinha, R., Askitis, N.: OzSort: Sorting 100GB for less than 87kJoules

[34] Zhao, J., Yun, Y.: A proximity language model for information retrieval. In: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval, ACM (2009) 291–298

[35] Deerwester, S., Dumais, S., Furnas, G., Landauer, T., Harshman, R.: Indexing by latent semantic analysis. Journal of the American society for information science **41**(6) (1990) 391–407

[36] Hofmann, T.: Probabilistic latent semantic indexing. In: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval, ACM New York, NY, USA (1999) 50–57

[37] Blei, D., Ng, A., Jordan, M.: Latent dirichlet allocation. The Journal of Machine Learning Research **3** (2003) 993–1022

[38] Teh, Y., Newman, D., Welling, M.: A collapsed variational bayesian inference algorithm for latent dirichlet allocation. Advances in Neural Information Processing Systems **19** (2007) 1353

[39] Minka, T., Lafferty, J.: Expectation-propagation for the generative aspect model. In: Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence, Citeseer (2002) 352–359

[40] Griffiths, T.: Gibbs sampling in the generative model of Latent Dirichlet Allocation. Standford University

[41] Griffiths, T., Steyvers, M.: Finding scientific topics (2004)

[42] Teh, Y., Jordan, M., Beal, M., Blei, D.: Hierarchical dirichlet processes. Journal of the American Statistical Association **101**(476) (2006) 1566–1581

[43] Blei, D., Lafferty, J.: A correlated topic model of science. Annals of Applied Statistics **1**(1) (2007) 17–35

[44] Blei, D., Griffiths, T., Jordan, M., Tenenbaum, J.: Hierarchical topic models and the nested Chinese restaurant process. Advances in neural information processing systems **16** (2004) 17–24

[45] Li, W., McCallum, A.: Pachinko allocation: DAG-structured mixture models of topic correlations. In: Proceedings of the 23rd international conference on Machine learning, ACM New York, NY, USA (2006) 577–584

[46] Mimno, D., Li, W., McCallum, A.: Mixtures of hierarchical topics with pachinko allocation. In: Proceedings of the 24th international conference on Machine learning, ACM New York, NY, USA (2007) 633–640

[47] Wallach, H.: Topic modeling: beyond bag-of-words. In: ACM International Conference Proceeding Series; Vol. 148: Proceedings of the 23 rd international conference on Machine learning, Association for Computing Machinery, Inc, One Astor Plaza, 1515 Broadway, New York, NY, 10036-5701, USA, (2006)

[48] Steyvers, M., Griffiths, T.: Latent semantic analysis: A road to meaning, chapter Probabilistic Topic Models. Laurence Erlbaum (2007)

[49] Wang, X., McCallum, A., Wei, X.: Topical n-grams: Phrase and topic discovery, with an application to information retrieval. In: Proceedings of the 7th IEEE International Conference on Data Mining. (2007) 697–702

[50] Griffiths, T., Steyvers, M., Blei, D., Tenenbaum, J.: Integrating topics and syntax. Advances in neural information processing systems **17** (2005) 537–544

[51] Gruber, A., Rosen-Zvi, M., Weiss, Y.: Hidden topic Markov models. In: Proceedings of International Conference on Artificial Intelligence and Statistics, Citeseer (2007)

[52] Blei, D., McAuliffe, J.: Supervised topic models. Advances in Neural Information Processing Systems **20** (2008) 121–128

[53] Mimno, D., McCallum, A.: Topic models conditioned on arbitrary features with Dirichletmultinomial regression. In: Proceedings of the. Volume 24.

[54] Titov, I., McDonald, R.: A Joint Model of Text and Aspect Ratings for Sentiment Summarization. Urbana **51** 61801

[55] Lacoste-Julien, S., Sha, F., Jordan, M.: DiscLDA: Discriminative learning for dimensionality reduction and classification. Advances in Neural Information Processing Systems 21 (NIPS08) (2008)

[56] Rosen-Zvi, M., Griffiths, T., Steyvers, M., Smyth, P.: The author-topic model for authors and documents. In: Proceedings of the 20th conference on Uncertainty in artificial intelligence, AUAI Press Arlington, Virginia, United States (2004) 487–494

[57] Wang, X., McCallum, A.: Topics over time: a non-Markov continuous-time model of topical trends. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM New York, NY, USA (2006) 424–433

[58] Newman, D., Chemudugunta, C., Smyth, P.: Statistical entity-topic models. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM New York, NY, USA (2006) 680–686

[59] Erosheva, E., Fienberg, S., Lafferty, J.: Mixed-membership models of scientific publications (2004)