Query by Document via a Decomposition-Based Two-Level Retrieval Approach

Linkai Weng[†‡], Zhiwei Li[‡], Rui Cai[‡], Yaoxue Zhang[†], Yuezhi Zhou[†], Laurence T. Yang[§], and Lei Zhang[‡]
[†]Dept. Computer Science and Technology, Tsinghua University.
[‡]Microsoft Research Asia.
[§]Dept. Computer Science, St. Francis Xavier University.
[†]wlk02@mails.tsinghua.edu.cn, [†]zyx@moe.edu.cn, [†]zhouyz@mail.tsinghua.edu.cn
[‡]{zli, ruicai, leizhang}@microsoft.com [§]ltyang@gmail.com

ABSTRACT

Retrieving similar documents from a large-scale text corpus according to a given document is a fundamental technique for many applications. However, most of existing indexing techniques have difficulties to address this problem due to special properties of a document query, e.g. *high dimensionality*, *sparse representation* and *semantic issue*. Towards addressing this problem, we propose a two-level retrieval solution based on a document decomposition idea. A document is decomposed to a compact vector and a few document specific keywords by a dimension reduction approach. The compact vector embodies the major semantics of a document, and the document specific keywords complement the discriminative power lost in dimension reduction process. We adopt locality sensitive hashing (LSH) to index the compact vectors, which guarantees to quickly find a set of related documents according to the vector of a query document. Then we re-rank documents in this set by their document specific keywords. In experiments, we obtained promising results on various datasets in terms of both accuracy and performance. We demonstrated that this solution is able to index large-scale corpus for efficient similarity-based document retrieval.

Categories and Subject Descriptors

H.3.1 [**Information Systems**]: Information Storage and Retrieval—*Content Analysis and Indexing*

General Terms

Algorithms, Performance, Experimentation

Keywords

document decomposition, similarity search, indexing

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGIR'11, July 24–28, 2011, Beijing, China.
Copyright 2011 ACM 978-1-4503-0757-4/11/07 ...$10.00.

1. INTRODUCTION

In many applications, a fundamental technique is to find similar documents according to a given document from a large-scale text corpus, e.g. a movie recommendation system needs to find similar movies/users to a given movie/user, in both content-based and collaborative filtering scenarios [30]; a blogger wants to know who pirates his blog pages or to do the cross reference; a patent lawyer want to find similar patents as reference cases. Although lots of retrieval technologies have been developed, most of them are designed for short query retrieval, e.g. 2-10 query terms. A document query, e.g. including 2000 query terms, is very different from a short query, as well as corresponding indexing techniques. Following [35], we term this problem as Query By Document (QBD) in this paper.

The QBD problem has three important properties, *high dimensionality*, *sparse representation* and *semantic issue*, which should be comprehensively considered in the solution. First of all, the document query contains lots of terms, so it is naturally a high-dimension indexing problem. Secondly, the document query is often represented as a language vector, which is quite sparse, and it will incur new challenges when mixed with the high dimension property. At last, a document query often has its inherent semantics, which is quite important information in the similar document matching. This property determines the difference between traditional near-duplicate detection and our QBD problem. The former often focus on the syntax level matching, and try to find exactly the same document only with some minor syntactic difference; the latter, needs to consider the syntactic level matching and semantic level matching together, and try to find related documents from the user perspective.

A simple solution for QBD is a linear scan approach, in which both documents in corpus and a query document are represented by high-dimensional vectors [3], and *cosine* similarity could be adopted to rank documents. However, this solution have two drawbacks: i) for large-scale corpus, its computational cost is unacceptable, ii) direct matching of keywords may encounter some language problems, i.e. *synonymy* and *polysemy* [13], which also loses the important semantic information.

Inverted files is an efficient index structure for large-scale text corpus [39]. In inverted files, words are primary keys, and documents containing the same word are organized as one row of the table. Thus, given a query, documents containing all query terms can be obtained efficiently. For a

typical query which contains 2-10 terms, this operation is very efficient [6]. However, inverted files cannot solve the QBD problem. Given a long document query, the computational cost to merge hundreds of rows of the table is unacceptable. Moreover, it is not necessary to require that the returned documents contain all terms of a query document. An optional solution is to select a set of representative words/phrases from the query document, and then search related documents via the inverted files with the constructed "short query" [35]. [35] proposed to score words by TF-IDF weights and mutual-information. However, the selection of representative words largely loses information of a query document, especially the semantic information. Moreover, inverted files cannot solve those language problems we mentioned above.

The QBD essentially is a special high-dimensional indexing (HDI) problem which has been extensively studied in literatures. A basic idea of HDI approaches is to partition a feature space to lots of "bins". If two documents appear in the same bin, they are deemed as similar documents. Typical HDI approaches can be categorized as tree-based, e.g. kd-tree [3], and hash-based approaches, e.g. locality sensitive hashing (LSH) [1]. Due to the well-known curse of dimensionality, tree-based approaches only can be applied to "low" dimensional spaces (say, less than 30) [22]. To find the nearest neighbors in a high-dimensional space by a tree-based approach, we have to perform back-tracking operation frequently in the search process, as will degenerate a such method to be a linear scan. LSH partitions a feature space by a group of carefully designed hash functions. It has been demonstrated to be an effective technique for visual features of images [20]. However, lots of experiments including ours show it is not good at indexing sparse feature vectors, e.g. TF-IDF vectors in QBD problem, because the L_2 distance of sparse features is not as reliable as that of dense features [1][1]. Most of hash-based approaches are designed for detecting near-duplicate documents [18], rather than retrieving similar documents, e.g. shingling algorithms [8] and a random projection-based approach [9].

Dimension reduction (DR) approaches are effective solutions to the QBD problem, e.g. matrix factorization based approaches (e.g. latent semantic indexing (LSI) [13], non-negative matrix factorization (NMF) [21]), and topic models (e.g. pLSI [19] and LDA [5]), in which high dimensional feature vectors are projected to a low dimensional latent space by minimizing some kinds of reconstruction error. Because the new representations of documents often are dense and compact, many similarity measures, e.g. *cosine* similarity, and HDI approaches, e.g. LSH, work well. Moreover, those language problems, i.e. *synonymy* and *polysemy*, which are often encountered in direct keywords matching-based solutions, can be alleviated [13].

However, DR approaches have an obvious drawback if they are applied to address QBD alone. DR approaches often reduce the discriminative power of document vectors. For example, two news stories, a basketball related and a football related, are likely to be projected to the same or two very near points in the latent space because "sports" are their major semantics (preserved in DR), while basketball and football are their minor semantics (ignored in DR), as shown in Figure 1. These minor semantics exist as residual which is abandoned in a DR process. This observation inspired us that the residual of documents can complement the loss of "characteristics" of documents. For a document, if we can selectively preserve some keywords, which major lead its residual, and along with the compact vector obtained by DR, we can preserve its major semantics as well as most of its discriminative power in a still compact compound representation.

With the above analysis, we proposed a decomposition-based two-level retrieval approach to address the QBD problem in this paper. Firstly, we utilize a principle document decomposition model(DDM)[10] to decompose the original document into three parts of words, i.e. background words, topic related words and document specific words. Background words have no information to distinguish documents, topic related words embody the inter-class differences which are major semantics of a document, and document specific words embody the intra-class differences which are minor but most distinguishing elements of a document. As shown in Figure 1, topic related words are major results of dimension reduction, while document specific words are residual. Then, we design an effective two-level index and rank schema based on the document decomposition results. In the indexing process, the compact and dense topic related vectors can be indexed by a HDI approach, e.g. LSH in this paper, and the document specific keywords can be indexed by inverted/forward files, as shown in Figure 3. In the ranking process, we first get a set of candidate documents by search in LSH index. Documents in this set are likely to talking about the same topics as the query. Then, we re-rank them by their document specific keywords. The top ranked documents are likely to talking about the same details of the same topics. [27] proposed a similar document decomposition idea with a model called PCP based on the PCA, which can also be included in our unified two-stage index and re-rank framework.

The remaining part of the paper is organized as follows. In Section 2, we introduce related works. In Section 3, we continue to introduce our document decomposition idea. Then the two-level index and rank schema is presented in Section 4. A set of practical experiments are conducted and the corresponding results are analyzed in section 5. Finally, conclusions and future work are given in the last Section.

2. RELATED WORKS

Near-duplicate detection is an important related work. It is a classical technique devoting to find duplicate pages or duplicate documents, which are strikingly similar with only some minute difference, e.g. words, fonts, bold types. Near-duplicates are widespread in email, retrieval results, news pages and images. The state-of-the-art solution is considered as the shingling algorithm in [8] and the random projection projection based approach in [9]. Moreover, there are lots of adaptive approaches to satisfy various requirement, such as [24, 11, 12, 14, 18], all of them gain great success in their own scenarios. As mentioned before, the goal of near-duplicate is not exactly the same with our QBD problems, in short, one focus on the syntactic similarity, and the other needs to consider semantic similarity and syntax similarity together. Locality sensitive hashing(LSH) [1] is frequently utilized in near-duplicate detection and is demonstrated effective in the

[1]Even if good parameters for sparse vectors exist, the process to finding them largely depends on designer's experience and extensive experiments.

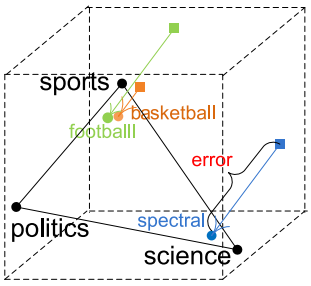

Figure 1: Illustration of the document decomposition idea. The cube denotes a vector space of documents, the triangle is a simplex spanned by three bases, i.e. topics, a rectangle point denotes a document, and a circle point denotes an embedding of a document in the topic simplex. The distance of a document to the simplex is its residual.

compact vector indexing, therefore, our approach adopt LSH to index the compact topic element of document.

Phrase extraction based matching is another related work to QBD problem, which tries to extract some representative phrases from the query document, and then to find related documents containing these representative phrases. Schemas to extract representative phrases are various: [25, 26] utilize statistical information to identify suitable phrases; [16, 33] incorporate the relationships between phrases to identify phrases; [32, 31] apply learning algorithms in the extracting process. [35] is the nowadays phrase extracting based matching approach, which identifies the phrase set by the TF-IDF weight and the mutual information, and then refines the set with the help of phrase association in the wikipedia link graph. Phrase extraction based matching is a quite effective solution to our problem, but the selection of phrases loses much useful information, especially the semantics in the document. In our approach, the document specific words actually are representative phrases which are extracted by a principle language model, and as a complement, the topic elements maintain the semantic information in document.

Topic model based retrieval is also a related work, which often combines topic model element and document language element together. This solution is quite popular nowadays, and [36, 37, 38, 23, 34] are all successful works of this class. The semantic information in document is somewhat complemented in the topic model element, so this solution has been demonstrated effective in many applications. However, there are still two challenges in these existing works: one is that the query is often seen as a bag of words, this assumption is reasonable in the short query scenario, but in our document query application, the semantic structure of the query also needs to be considered; the other one is that when facing the document query scenario with large-scale corpus, the matching efficiency will be a serious issue. To address these two challenges, our approach proposes the document decomposition idea and two-level indexing schema.

3. DOCUMENT DECOMPOSITION

In this section, we present the document decomposition idea and its implementation. It is noted the purpose of decomposing a document is to find a new representation which can be easier indexed.

3.1 Document Decomposition Idea

Our document decomposition idea is originated from dimension reduction approaches. To make the presentation easy to understand, we use LSI to illustrate our basic idea [13]. By LSI, a document, d (a W-dimensional sparse vector, where W is the size of vocabulary), is decomposed to

$$d = \mu + Xw + \epsilon \tag{1}$$

where μ is the mean of features, X is a $W \times k$ matrix, k is the number of principle components ($k \ll W$ for dimension reduction puspose), w is a k-dimensional coefficient vector, and ϵ is a W-dimensional vector. Column vectors of X is a group of so-called base vectors, which span a low-dimensional latent space; w is the projection of a document in the space; ϵ is the residual which is the "distance" of the document to the space. Since, the mean vector, μ, does not contain any discriminative information (i.e. stopwords), it can be removed from the representation of a document. After removing μ, the inner product of a document d in corpus and a query document q could be computed by

$$< d, q >=< Xw_d + \epsilon_d >< Xw_q + \epsilon_q >= w_d^T w_q + \epsilon_d^T \epsilon_q \tag{2}$$

Here, we use a property of LSI decomposition: the residual (ϵ) is orthogonal to principle components (X). Thus, the multiplication of X and ϵ is null. Considering the physical meanings of w and ϵ, we can get an effective approximation to the inner product. For a good LSI decomposition, entries of ϵ usually are very close to zero. Thus, even if we only keep a few large entries of the two ϵ vectors respectively, the inner product of two documents is likely to not changing. However, in this way, we can greatly reduce the storage for documents. To store a raw document vector, d, we need $|d|$ storage cells, where $|d|$ is the length of the document, while to store the new appropriate representation, we only need $k + t$ storage cells, where t is the number of ϵ entries we kept. $k + t$ is much less than $|d|$ in practice. Moreover, in Section 4, we will show that a good property of this approximation enables an efficient indexing solution. In summary, a document is represented by:

$$d \triangleq \{w, \varphi\} \tag{3}$$

where w is a k-dimensional compact vector, and φ is a few keywords.

It is noted that, for short documents, the new representations may increase their storage cost (i.e. $k + t > |d|$). However, we argue that even in this case, the proposed approach has benefits. For a short document, the information containing in it is likely to be insufficient if we only consider words in it alone. In this case, LSI can find complementary information for short documents [29, 34] because they consider all words in the whole corpus. Relationships among words can help short documents to complement their information.

Actually, the document decomposition idea is very general. Any dimension reduction approach, if only it can find the most significant residual words explicitly, can be adopted to implement the idea, e.g. the LSI approach presented above. However, we recommend to implement it by a probabilistic topic model in this paper. Although the computations in the topic model looks much more complex than

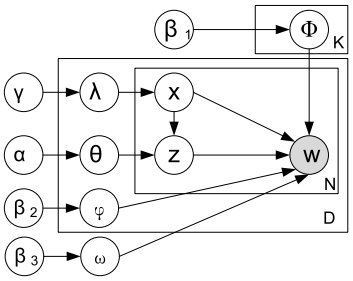

Figure 2: A graphical representation of document decomposition model (DDM).

matrix factorization-based approaches, it can help us better understand the proposed solution[2]. Indeed, the two approaches are equivalent. [15] proved the equivalence of pLSI [19] and non-negative matrix factorization [21].

3.2 A Graphical Model Implementation

A topic model often describes a generative process of a document [5, 19]. It assumes that words of a document are drawn from a set of topic distributions. Document decomposition model (DDM) [10], as shown in Figure 2, is an extension of LDA [5], which is first proposed to model different aspects of a document. The DDM model assumes a document is generated by this process:

1. Draw $\lambda \sim Dirichlet(\gamma)$
2. Draw $\varphi \sim Dirichlet(\beta_2)$
3. Draw $\theta \sim Dirichlet(\alpha)$
4. For each word of the document
   (a) Draw a $x \sim Mult(\lambda)$
   (b) if $x = 1$, draw $z \sim Mult(\theta)$, and then draw a word $w \sim Mult(\phi_z)$
   (c) if $x = 2$, draw a word $w \sim Mult(\varphi)$
   (d) if $x = 3$, draw a word $w \sim Mult(\omega)$

where $Mult$ stands for a multinomial distribution. The key idea of DDM is to use a switch variable x to control the generation of words. x takes value 1, 2 or 3, which controls a word is drawn from either a topic distribution (z and ϕ_z), a document specific distribution (φ), or a corpus background distribution (ω). It provides a nature way to partition a document into three parts of words. Intuitively, a word which widely appears in all documents is likely to be a background/stop word; a word which only appears in a few documents but seldom appears in other documents is likely to be a document specific word; a word which widely appears in documents with a common semantic but seldom appears in other documents without this semantic is likely to be a topic related word. It is interesting to note that document specific words are likely to have the largest TF-IDF values in a document.

From the aspect of matrix factorization, the work of DDM can be understood as simultaneously finding a group of basis vectors ($\phi_{1:K}$, a set of distributions over vocabulary), and coefficients (θ, topic mixture proportion). The conditional probability of a word given a document is computed by

$$p(w|d) = \sum_z p(w|z)p(z|x=1,d)p(x=1|d) + p(w|x=2,d)p(x=2|d) + p(w|x=3,d)p(x=3|d)$$

Thus, the decomposition can be shown in a matrix factorization manner

$$d = \begin{bmatrix} . & . & . \\ . & \phi_{ij} & . \\ . & . & . \end{bmatrix} \begin{bmatrix} . \\ \theta_d \gamma_{d1} \\ . \end{bmatrix} + \begin{bmatrix} . \\ \varphi_d \\ . \end{bmatrix} \gamma_{d2} + \begin{bmatrix} . \\ \omega \\ . \end{bmatrix} \gamma_{d3}$$

This decomposition has the same format as Eq. 1.

3.2.1 Parameter Estimation

To estimate parameters of a graphical model, Expectation Maximization (EM) algorithm is often adopted [19, 5]. However, in DDM, direct optimization to model parameters by EM algorithm is intractable [5, 10]. We adopt Monte Carlo EM [2] instead. In E step, rather than analytically compute the posterior of hidden variables, we draw samples from their posterior distributions. In M-step, we approximate the expectation by a finite sum over samples and maximize the expectation with respect to model parameters. Instead of uisng a full Markov Chain Monte Carlo (MCMC) approach [2], a collapsed Gibbs sampling approach is used to draw samples from posterior [17]. Considering the conditional independencies between variables, we only need to sample hidden variables x and z, while integrate out other hidden variables. The Gibbs sampling equations are easy to obtain

$$p(x_i = 1, z = k|x_{-i}, z_{-i}, w, \Theta) = \frac{\gamma_1 + n^{1,\cdot}_{d,-i}}{\sum_{j=1}^3 \gamma_j + n^{j,\cdot}_{d,-i}} \times \frac{\alpha_k + n^{1,k}_{d,-i}}{\sum_{j=1}^K \alpha_j + n^{1,j}_{d,-i}} \times \frac{\beta_{1,w_i} + n^{1,k}_{-i,w_i}}{\sum_{j=1}^W \beta_{1,j} + n^{1,k}_{-i,j}}$$

$$p(x_i = 2|x_{-i}, z_{-i}, w, \Theta) = \frac{\gamma_2 + n^{2,\cdot}_{d,-i}}{\sum_{j=1}^3 \gamma_j + n^{j,\cdot}_{d,-i}} \times \frac{\beta_{2,w_i} + n^{2,w_i}_{d,-i}}{\sum_{j=1}^W \beta_{2,j} + n^{2,\cdot}_{d,-i}}$$

$$p(x_i = 3|x_{-i}, z_{-i}, w, \Theta) = \frac{\gamma_3 + n^{3,\cdot}_{d,-i}}{\sum_{j=1}^3 \gamma_j + n^{j,\cdot}_{d,-i}} \times \frac{\beta_{3,w_i} + n^{3,w_i}_{\cdot,-i}}{\sum_{j=1}^W \beta_{3,j} + n^{3,j}_{\cdot,-i}}$$

where Θ denotes all hyper parameters, i.e. $\alpha, \gamma, \beta_{1,2,3}$, $-i$ means all words except for the current word w_i, $n^{j,k}_{d,-i}$ denotes the number of words generated when $x = j$ and $z = k$ in document d, $n^{j,w_i}_{d,-i}$ denotes the number of times of word w_i generated when $x = j$ in document d, and $n^{1,k}_{-i,w_i}$ denotes the number of times of word w_i assigned to $z = k$. To update hyper-parameters in M step, we adopt the fix-point iteration algorithm proposed in [28].

3.2.2 Inference for Unseen Documents

In practice, we only can learn a model with a small proportion of documents of a large corpus, and then apply it to infer hidden variables of remainder documents. For a retrieval system, we also need to infer latent variables of a query document. For DDM, the inference algorithm is similar as the estimation algorithm. However, model parameters will be fixed to be those obtained in model estimation step. The Gibbs sampling equations are

$$p(x_i = 1, z = k|x_{-i}, z_{-i}, w, \Theta) = \frac{\gamma_1 + n^{1,\cdot}_{d,-i}}{\sum_{j=1}^3 \gamma_j + n^{j,\cdot}_{d,-i}} \times \frac{\alpha_k + n^{1,k}_{d,-i}}{\sum_{j=1}^K \alpha_j + n^{1,j}_{d,-i}} \times \phi_{k,w_i}$$

$$p(x_i = 2|x_{-i}, z_{-i}, w, \Theta) = \frac{\gamma_2 + n^{2,\cdot}_{d,-i}}{\sum_{j=1}^3 \gamma_j + n^{j,\cdot}_{d,-i}} \times \frac{\beta_{2,w_i} + n^{2,w_i}_{d,-i}}{\sum_{j=1}^W \beta_{2,j} + n^{2,\cdot}_{d,-i}}$$

$$p(x_i = 3|x_{-i}, z_{-i}, w, \Theta) = \frac{\gamma_3 + n^{3,\cdot}_{d,-i}}{\sum_{j=1}^3 \gamma_j + n^{j,\cdot}_{d,-i}} \times \omega_{w_i}$$

[2]The computational cost of the two approaches are almost the same.

It is remarkable that the inferences of latent variables of different documents are independent to each other. From above equations, we can easily verify this fact. Only statistics inside a document take part in the computation of Gibbs sampling, that is, we can distribute documents to different processors and parallel decompose them. Thus, the scalability of DDM is not an issue.

Once we have inferred latent variables of a document, we can easily get its decomposition results. θ is the compact vector as the vector obtained in the LSI-based approach. To find out document specific words of a document, we compute this conditional probability

$$p(x_i = 2|w_i, d) = \frac{p(w_i|x_i = 2, d)p(x_i = 2|d)}{p(w_i|d)} \propto \frac{\varphi_{d,w_i}}{p(w_i|d)}$$

To reduce computation, we approximate the posterior by φ_{d,w_i}. With this probability, we can control the number of document specific words to be an acceptable value, e.g. 15 in our experiments. An advantage of DDM beyond matrix factorization-based approaches is that it can explicitly decompose words of a document to be three categories. This is very helpful when we analyze experimental results.

3.3 Other Implementations

So far we have proposed two mathematically solid approaches to implement the document decomposition idea, but actually we can implement it by many heuristic ways. The only requirement to an approach is that it must be able to get a compound representation of a document like in Eq. 3. We give some examples which will be compared in our experiments.

Under the framework of **topic models**, we need to construct a set of topics($\phi_{1:K}$ of DDM), and then run the inference algorithm of DDM to decompose documents. For this purpose, we can obtain topics by many existing approaches, such as LDA [5] and pLSI [19]. A heuristic but very practicable method is to get topic vectors from web pages of ODP by counting word frequencies in documents of each category. An ODP category may be mapped to a topic or a few topics. These topic vectors are likely to be more reasonable than those got by purely unsupervised models.

Under the framework of **matrix factorization**, we can perform the factorization in many methods, e.g. LSI [13] and NMF [21]. If only we can get a decomposition like Eq. 1, we can extract the co-efficient w as a representation of topic related words and keep the top N largest entries of ϵ as document specific words.

4. INDEXING AND RANKING

We have represented both documents in corpus and a query document by vector pairs $<\theta, \varphi>$, in which θ is a compact vector and φ is a few keywords. In this section, we will present indexing and ranking approaches based on this compound representation.

4.1 Ranking Function

The similarity of two documents is computed by a linear combination of the two components:

$$sim(d, q) = \gamma_{d1}\gamma_{q1}sim(\theta_d, \theta_q) + \gamma_{d2}\gamma_{q2}sim(\varphi_d, \varphi_q) \quad (4)$$

where $\gamma_{.1}$ and $\gamma_{.2}$ mean the word ratios of topic related words and document specific words in a document respectively. They can be got by the inference algorithm. In experiments, we compute $sim(\theta_d, \theta_q)$ and $sim(\varphi_d, \varphi_q)$ by inner products. This simple ranking function does not introduce additional parameters.

If we keep all words in φ (φ is equivalent to the full residual ϵ in Eq. 2), this similarity measure exactly recovers the Eq. 2. Thus, this ranking approach is named as "accurate ranking" in this paper. It defines the upper-bound of retrieval accuracy of the compound representations. In following sections, we will develop an index-based approximation to it, which is required to be as accurate as it but with much lower computational and memory cost.

Similar as Eq. 4, [34] linearly combines probabilities obtained by a traditional language model and a topic model, which achieved best performance in lots of short query retrieval tasks. Without changing its meanings, its ranking function can be rewritten as

$$p(q|d) = \lambda \sum_{w \in q} p(w|d) + (1 - \lambda) \sum_{w \in q} \sum_{z} p(w|z)p(z|d) \quad (5)$$

where z denotes a topic in a LDA model and λ is a coefficient to balance the two terms. By simple derivation, we can prove this function is equivalent to Eq. 4 when the probability $p(w|q) = 1/|q|$ and $p(z|q) = 1/K$. That is, it assumes that the conditional probabilities of different words/topics given the query q are equal. This assumption is reasonable for a short query because we do not have enough information to infer hidden variables only with a few terms. However, for a real document query, we have enough information to infer hidden variables to extract more knowledge from it. Thus, our work can be deemed as an extension of [34] to dealing with long document queries.

4.2 Indexing Scheme

By fully considering the characteristics of the new representations of documents, we designed an efficient indexing structure as shown in Figure 3. We build two separate indices for topic related words (the compact vector θ) and document specific words (φ). The two index are used to approximately compute the similarity given in Eq. 4.

Indexing topic related words. Because topic related words of a document are represented by a compact and dense vector, which is an appropriate data style for LSH, we choose LSH [1] to index them. Previous research [20] and our experiments demonstrated the LSH algorithm is good at indexing compact and dense feature vectors but not good at indexing high-dimensional sparse feature vectors, like TF-IDF features of documents. LSH algorithm will assign a group of hash values to a document. These hash values can be deemed as a new discrete representation of a document. We further adopt inverted list to index them to enable fast similarity computing [20]. Therefore, the memory cost of LSH index is only $L \times D$, where L (e.g. 30 in our experiments) is the number of hash functions of LSH.

Indexing document specific words. Because we can select the most salient document specific words according to the conditional probability $p(x_i = 2|w_i, d)$, we can control the number of document specific words. In experiments, we achieved good retrieval accuracy with only 15 document specific words. Thus, we adopt a simple structure to organize document specific words, i.e. forward list. In the forward list, each entry is a pair $<$*word id*, *weight*$>$, in which *weight* is φ_w. Therefore, the memory cost of the forward list index is only $15 \times D$, where D is the number of documents.

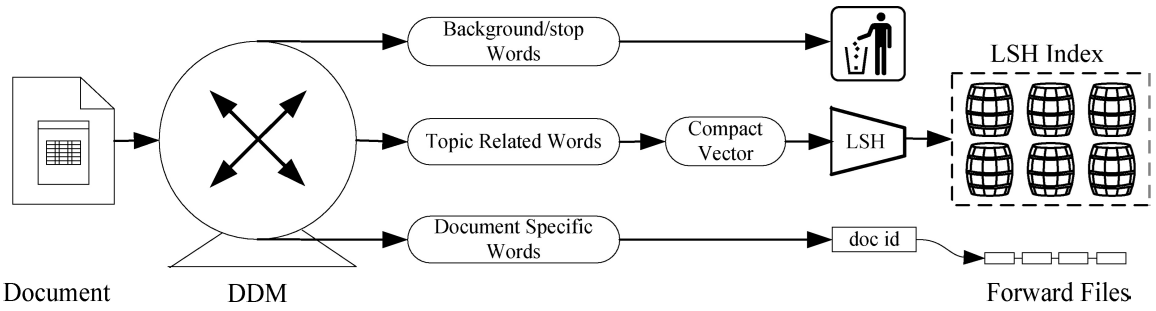

Figure 3: A workflow of our index scheme. The "X" in the centrifuge module motivates the code name of this project, DocX. A centrifuge machine uses centrifugal force for separating substances of different densities, for removing moisture, or for simulating gravitational effects.

4.3 Two-Level Ranking Approach

The functionalities of the two components of the compound representations imply an efficient two-level ranking approach. The similarity $sim(\theta_d, \theta_q)$ distinguish documents belonging to different categories, while $sim(\varphi_d, \varphi_q)$ only is a complement of the first similarity which can further distinguish documents belonging to the same category. Thus, the first similarity is a major component, and the second similarity is a minor component. This observation naturally motivated a two-level ranking approach. In the first stage, we use $sim(\theta_d, \theta_q)$ to select the top k most similar documents as a candidate set. Because we index θ by LSH, we can get a set of related documents much more quickly than a linear scan approach. In the second stage, we use $sim(\varphi_d, \varphi_q)$ to rerank documents in this set. Actually, multiple-level ranking approaches are very popular in IR systems [7].

In our experiments, we set the size of candidate set to 50. The time complexity to respond a query is $30 \times \#collision + 15 \times 50$, where $\#collision$ is the average number of documents in the same bucket in inverted index of LSH. In LSH, $\#collision$ only slightly changes with the increasing of the numbers of documents it indexes [1]. Thus, the solution can answer a query almost in constant time, which is independent to the number of documents the system indexes.

It is noted that, by this index and ranking approach, we only can get an approximate results as the accurate ranking(Eq. 4) because: i) LSH is an approximate nearest-neighbor algorithm; ii) only a few most salient document specific words are kept in index. Although it is an approximate algorithm, its retrieval accuracy is almost as good as the linear ranking in our experiments. The small sacrifice of accuracy significantly shortens the response time and reduces memory cost.

4.4 Scalability

To the best of our knowledge, there are no widely acknowledged criterions on what is a scalable indexing solution. However, the following aspects must be useful when checking the scalability of an indexing solution.

Parallelizable. It implies the indexing of documents must be independent, and its ranking function only uses features within a document and some simple global statistics (e.g. PageRank [6]).

Efficient. The time to building index (e.g. in a few hours) and responding users' queries (e.g. in a few milliseconds) must be acceptable.

Large Capability. Single machine can index several million documents (e.g. Google indexed 5 million pages by one machine in 2003 [4]).

It is easy to check that our indexing and ranking solution can meet these requirements. In our experiments, it takes about 8 hours to decompose 5 million documents and build index by one 3G CPU. The index size of 5 million pages is about 3.5GB (including LSH index, forward files and necessary metadata of documents). The average time to answer a query is about 13ms if the top 50 results are returned. Thus, theoretically, the proposed solution can index any number of documents and provide realtime search service.

5. EXPERIMENTS

We now present the experiment evaluation of our approach. Section 5.1 and Section 5.2 introduce the utilized data sets and the related experiment setup respectively. Section 5.3 evaluates performance of decomposition based approaches, and Section 5.4 evaluates performance of the indexing solution. At last, the scalability issues are discussed in the Section 5.5 to show the efficiency of our approach.

5.1 Data sets

We evaluated the proposed approaches on four datasets shown in Table 1: 20 newsgroups (20NG), Wikipedia pages (Wiki), a randomly selected subset of Wikipedia (SWiki) and a web corpus with 5 million pages (5M) which was randomly sampled from the web TREC'08 corpus. In 20NG, documents belonging to the same category are labeled as relevant documents. Wikipedia organizes documents in a tree structure. If two leaf nodes, i.e. documents, have the same parent or are linked by a "see also" hyperlink, they are labeled as similar documents. SWiki dataset is used to investigate the affection of corpus size to our approaches. For the 5M dataset, we do not have ground truth. The purpose of collecting this dataset is to evaluate scalability of our approaches. In the preprocessing, we performed Porter's stemming and removed words whose document frequencies (DF) are less than 4.

5.2 Experimental Setup

We implemented four baseline approaches: TF-IDF with *cosine* similarity (TFIDF) [3], directly indexing TF-IDF vectors by LSH (LSH) [1], similarity computed only by top N TF-IDF words(TopN) [35], and linearly combining similarity computed by top N TF-IDF words and similarity

Table 1: Details of the four dataset

Dataset	#Train	#Test	#Terms
20NG	11,314	7,532	54,071
Wiki	60,000	1.44 Million	113,748
SWiki	20,000	88,204	98,046
5M	100,000	5 Million	163,871

got by topic mixtures obtained by LDA (TopNLDA) [34]. TopNLDA is one of the state-of-the-art algorithms for web search and also an instantiation of the proposed document decomposition idea. To simplify the notation, we refer to document specific words as DS, and topic related words as TS. We refer to Eq. 4 as DDM, and our index scheme as DDM+Index. We randomly sampled 200 documents from each dataset as queries. In the following experiments, we will adopt the macro-average precision of the top 10 search results for the 200 queries to measure algorithms' performance, which is computed by:

$$\Pr@10 = \frac{1}{N}\sum_{i=1}^{N}\frac{T_i}{10}$$

where N is the number of queries and T_i is the number of related documents in the top 10 search results for the *ith* query.

Our experimental goals include three-fold: 1) performance of the DDM and some practical implementations (rank by Eq. 4); 2) the drop of performance of the index and ranking solution; 3) scalability of the solution.

5.3 Performance of Decomposition-Based Approaches

In this section, we evaluate the best performance that decomposition-based approaches can obtain, which is achieved by Eq. 4.

5.3.1 Results with Different Number of Topics

A common concern about a topic model based algorithm is that the number of topics may significantly affect its performance. How to determine the number of topics is still an open problem. Thus, we run a group of experiments to investigate the effects of topic numbers. In these experiments, we fixed the number of DS words to 15. The results on the three dataset are shown in Figure 4.

The decomposition based approaches significantly outperformed baseline approaches. On the three dataset, the LSH approach almost consistently got the worst results. This fact indicates that LSH is not a good approach to index high-dimensional sparse feature vectors. The accuracy of TS only approach increases with the number of topics, while the accuracy of DS only approach drops. Because in this process more and more DS words will be absorbed into topics, the TS is becoming more and more powerful, while the DS is becoming weaker and weaker. In extreme cases, both DS and TS only approaches degenerate to the TF-IDF approach.

Intuitively, the best results of an approach on the three dataset will be obtained under very different number of topics because the complexities of them are very different. However, we found all their best results were achieved when topic numbers are around 250. The result may be explained by the way we obtain TS and DS words. There are no criterions to define what is a TS word but not a DS word and vice versa. If we increase the number of topics in DDM, some words which previously are classified as DS words will be re-labeled as TS words, and vice versa. Thus, for a dataset, if it can be well represented by the latent space, DS words could be very discriminative or rare words which represent minor semantics of documents, otherwise, DS words will be more like TS words which represent major semantics of documents. Thus, the proposed decomposition-based approaches are not sensitive to dataset or number of topics.

All approaches based on the document decomposition idea outperform the baseline approaches. As we have mentioned, words with largest TF-IDF values are likely to be DS words. However, the TopNLDA got worse accuracy than decomposition based approaches. It is probably because the top N TF-IDF words may be redundant to the topics obtained by LDA, while DDM can decompose a document to two sets of words which are complementary to each other. Moreover, the accuracy of ODP+DS, is close to DDM. This fact further demonstrates the power of the document decomposition idea. Because curves for the two other DDM-based approaches (LDA+DS and LSI+DS) almost overlapped with ODP+DS and DDM, we did not draw them in this figure.

5.3.2 Results with Different Number of DS Words

Theoretically, the number of DS words should be different for different kinds of documents. For example, Wikipedia documents usually are long documents with complex semantics, while documents of 20NG usually are short and simple documents. Thus, DS words for Wikipedia documents should be more than 20NG. However, after we performed experiments we got different answers which are shown in Figure 5. By setting the number of topics to be 250 and the number of DS words to be 15, almost all approaches significantly outperformed baseline approaches. Even when we set the number of DS words to be 5, their performances are better than TopN. The accuracies of TopNLDA is worse than DDM based approaches again. This result could be explained by the reason we discussed in last section. It again demonstrates that DS words are more meaningful than top N TF-IDF words in this ranking framework. Thus, we conclude that 250 topics and 15 DS words are an appropriate setting for a QBD problem.

5.4 Performance of the Indexing Solution

In this section, we evaluate performance of the index solution.

5.4.1 Results with Different Number of Topics

We built index for documents according to the proposed indexing scheme and adopted the two-level ranking approach to rank documents. Because we only evaluate the precision of the top 10 search results, we set the size of candidate set to be 50 in the LSH retrieval step. LSH index has two parameters, i.e. L and k. L is the number of hash functions, which is set to 30; k is the length of hash code, which is set to 6. The results are shown in Figure 6. For all index-based approaches, their curves are only slightly lower than corresponding accurate approaches (the DDM curve). We assert that the two approaches, i.e. DDM and DDM+index, are statistically the same. We run t-test to evaluate this hypothesis. With 99.5% confidence, the hypothesis is true. Therefore, we conclude that the indexing solution is a good approximation to the accurate approaches.

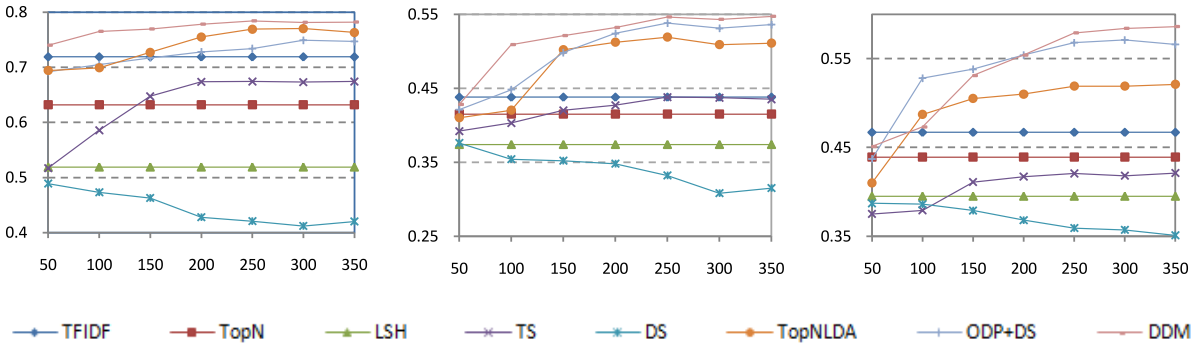

Figure 4: Macro-average Pr@10 change with the number of topics on 20NG, Wiki, and SWiki. TS means that only topic mixtures are used to compute document similarity, DS means that only DS words are used to perform retrieval, and ODP means that topics are got by counting ODP pages.

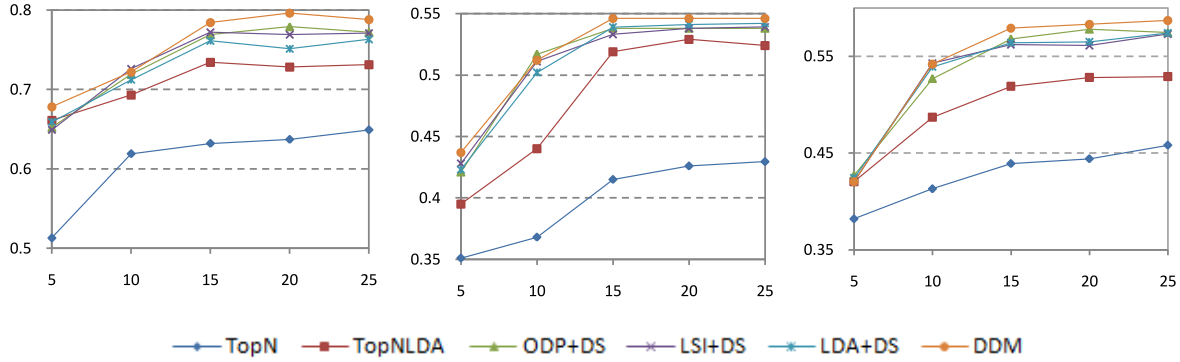

Figure 5: Macro-average Pr@10 change with the number of document specific words on 20NG, Wiki, and SWiki. LSI and LDA mean that topics are obtained by LSI decomposition (3000 documents were used to perform LSI decomposition) and LDA model, respectively

5.4.2 *Results with Different Number of DS Words*

Figure 7 shows the results with different number of DS words. Curves of DDM and DDM+index almost overlap together at all points. When the number of DS words is bigger than 10, the accuracies of all our approaches are very close. These results further demonstrate that the index solution is a good approximation to the accurate approach.

Combining results of these experiments, we can conclude: 1) DDM outperform other baseline approaches; 2) accuracies of the proposed practical approaches (e.g. ODP+DS) are only slightly worse than DDM; 3) The drop of accuracy of the index solution is acceptable.

5.5 Scalability Issues

The efficiency of a retrieval system could be measured from two-fold, i.e. time and memory cost to build index and answer a query. In our system, a standard index machine is a workstation with a dual-core CPU and 4G bytes RAM, by which we plan to index around 5 million web pages and provide realtime search service. Thus, we mainly measure cost of our approaches under this hardware setting.

5.5.1 *Cost of building index*

In our system, building index consists of three steps: train a model by a set of documents, infer the left documents and build index as the proposed indexing scheme. The model training is very time-consuming on a large corpus with huge lexicon. It took us about 50 hours to train a model with 100,000 documents (its vocabulary consists of 163,871 terms). Fortunately, we can get the model done once and for ever. The inference time of the three dataset are listed in Table 2. The time for long documents (e.g. Wiki) are slightly longer than short documents (e.g. 20NG). The time per document on all dataset is no longer than 4.5ms. As we have mentioned, the inference could be performed parallel. By a dual-core computer, it takes around 3.2 hours to infer 5 million documents. After we get inferring results, index can be built in around 70 minutes.

Table 2: Inference time for three dataset (#topics=250)

Dataset	#Docs	#Terms	Avg. Time(ms)
20NG	7,532	54,071	3.53
Wiki	1.44 Million	113,748	4.38
5M Pages	5 Million	163,871	4.213

In a real system, index should be able to be loaded in memory to enable fast search [6, 4]. The LSH index size of the 5M (#topics=250, LSH parameters L=30 and k=6 [1, 20]) is around 2.1GB. We use a <*word id*, *weight*> pair to represent a document specific word in the forward files. The size of forward files and lexicon of the 5M is around 800MB. Thus, all index data can be loaded in memory.

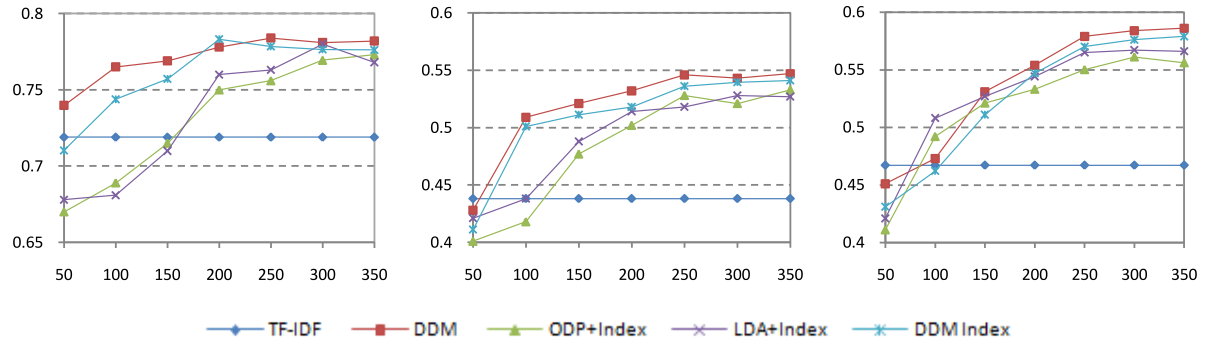

Figure 6: Macro-average Pr@10 of index scheme change with the number of topics on 20NG, Wiki, and SWiki

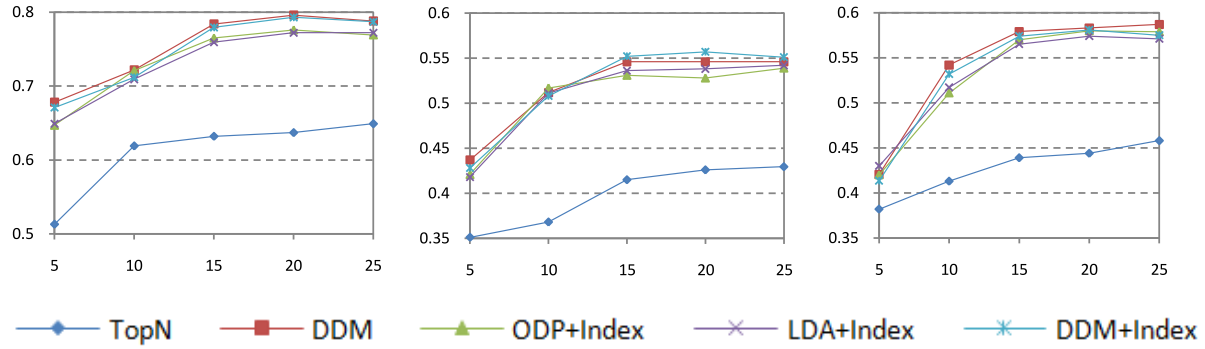

Figure 7: Macro-average Pr@10 of index scheme change with the number of document specific words on 20NG, Wiki, and SWiki

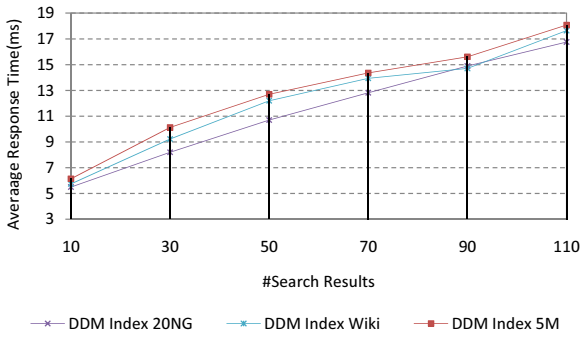

Figure 8: Response time under different size of index

5.5.2 *Time to answer a query*

The average response time as a function of number of returned search results is shown in Figure 8. The response time includes the time to inferring a query document, i.e. about 4ms, and the time for ranking. The response time is almost a constant time, which almost does not increase with the number of documents that the system indexes. If we run accurate ranking algorithm on the 5M pages dataset, the time to answer a query is around 25s. The proposed index solution can accelerate the search speed 100-1000 times, while its results are almost as accurate as the "accurate" ranking approach (Eq. 4). Our system is able to answer more than 20 queries per second.

6. CONCLUSION

We have proposed a solution for large-scale query by document (QBD) application, which essentially is a fast approach to compute nearest neighbors in a high-dimensional sparse feature space. Our theoretic analysis and experimental results demonstrate:

1) The document decomposition idea can solve problems caused by long document queries. Based on this idea, we can represent a document to be a compact vector as well as a few keywords. Retrieval results based on the new representation are better than full text linear scan approach.

2) The indexing and two-level ranking solution significantly reduces both computation and memory cost of the system, while it can provide search results as accurate as the accurate ranking algorithm (Eq. 4). The whole indexing solution is a good trade-off among accuracy, cost and performance.

3) The solution has the same architecture as web search engines. Thus, it can be applied to index large-scale text corpus to support QBD-based applications.

7. REFERENCES

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

[2] C. Andrieu, N. de Freitas, A. Doucet, and M. I. Jordan. An introduction to mcmc for machine learning. *Machine Learning*, 50(1-2):5–43, 2003.

[3] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[4] L. A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.

[5] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *JMLR*, 2003.

[6] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[7] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *CIKM*, 2003.

[8] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Comput. Netw. ISDN Syst.*, 29(8-13):1157–1166, 1997.

[9] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.

[10] C. Chemudugunta, P. Smyth, and M. Steyvers. Modeling general and specific aspects of documents with a probabilistic topic model. In *NIPS*, 2006.

[11] J. Cho, N. Shivakumar, and H. Garcia-Molina. Finding replicated web collections. *SIGMOD Rec.*, 29(2):355–366, 2000.

[12] A. Chowdhury, O. Frieder, D. Grossman, and M. C. McCabe. Collection statistics for fast duplicate document detection. *ACM Trans. Inf. Syst.*, 20(2):171–191, 2002.

[13] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *JASIS*, 41(6):391–407, 1990.

[14] F. Deng and D. Rafiei. Approximately detecting duplicates for streaming data using stable bloom filters. In *SIGMOD '06*, pages 25–36, New York, NY, USA, 2006. ACM.

[15] C. H. Q. Ding, T. Li, and W. Peng. Nonnegative matrix factorization and probabilistic latent semantic indexing: Equivalence chi-square statistic, and a hybrid method. In *AAAI*, 2006.

[16] K. T. Frantzi. Incorporating context information for the extraction of terms. In *ACL'97*, pages 501–503, Morristown, NJ, USA, 1997. Association for Computational Linguistics.

[17] T. L. Griffiths and M. Steyvers. Finding scientific topics. *Proc Natl Acad Sci U S A*, April 2004.

[18] M. R. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *SIGIR*, 2006.

[19] T. Hofmann. Probabilistic latent semantic indexing. In *SIGIR*, pages 50–57, 1999.

[20] P. Indyk and N. Thaper. Fast image retrieval via embeddings. In *International Workshop on Statistical and Computational Theories of Vision*, 2003.

[21] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *NIPS*, 2000.

[22] T. Liu, A. W. Moore, and A. G. Gray. New algorithms for efficient high-dimensional nonparametric classification. *Journal of Machine Learning Research*, 7:1135–1158, 2006.

[23] X. Liu and W. B. Croft. Cluster-based retrieval using language models. In *SIGIR '04*, pages 186–193, New York, NY, USA, 2004. ACM.

[24] D. P. Lopresti. Models and algorithms for duplicate document detection. In *ICDAR '99*, page 297, Washington, DC, USA, 1999. IEEE Computer Society.

[25] C. D. Manning and H. Schutze. *Foundations of Statistical Natural Language Processing*. The MIT Press, June 1999.

[26] O. Medelyan and I. H. Witten. Thesaurus based automatic keyphrase indexing. In *JCDL '06*, pages 296–297, New York, NY, USA, 2006. ACM.

[27] K. Min, Z. Zhang, J. Wright, and Y. Ma. Decomposing background topics from keywords by principal component pursuit. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM '10, pages 269–278, New York, NY, USA, 2010. ACM.

[28] T. P. Minka. Estimating a dirichlet distribution. Technical report, Microsoft, 2000.

[29] X. H. Phan, M. L. Nguyen, and S. Horiguchi. Learning to classify short and sparse text & web with hidden topics from large-scale data collections. In *WWW*, 2008.

[30] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *WWW'01*, 2001.

[31] T. Tomokiyo and M. Hurst. A language model approach to keyphrase extraction. In *Proceedings of the ACL 2003 workshop on Multiword expressions*, pages 33–40, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[32] P. D. Turney. Learning algorithms for keyphrase extraction. *Inf. Retr.*, 2(4):303–336, 2000.

[33] O. Vechtomova and M. Karamuftuoglu. Approaches to high accuracy retrieval: Phrase-based search experiments in the hard track. In *TREC*, 2004.

[34] X. Wei and W. B. Croft. Lda-based document models for ad-hoc retrieval. In *SIGIR*, pages 178–185, 2006.

[35] Y. Yang, N. Bansal, W. Dakka, P. Ipeirotis, N. Koudas, and D. Papadias. Query by document. In *WSDM'09*, 2009.

[36] C. Zhai and J. D. Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *CIKM*, pages 403–410, 2001.

[37] Y. Zhang, J. P. Callan, and T. P. Minka. Novelty and redundancy detection in adaptive filtering. In *SIGIR*, pages 81–88, 2002.

[38] Y. Zhang, W. Xu, and J. Callan. Exact maximum likelihood estimation for word mixtures. In *In Text Learning Workshop in ICML'02*, 2002.

[39] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2):6, 2006.