Query by Documents on Top of a Search Interface

Nhat X.T. Le*, Moloud Shahbazi, Abdulaziz Almaslukh, Vagelis Hristidis

Department of Computer Science & Engineering, University of California, Riverside
900 University Ave., Riverside, CA, 92521 United States

Abstract

Document repositories often provide a keyword-based query interfaces to allow users to search for documents. These interfaces typically have rate limits or monetary cost per access operation. Constrained search interfaces include legal or medical data sources, social networks and the Web. We study the problem where a user has a set of input documents, and wants to discover other similar documents using a constrained search interface. Specifically, given a set of input documents and an access budget, we present principled techniques to generate a list of queries to submit. Our technique's key intuition is to compute the best set of queries to return the input documents, which, as we show experimentally, also return other relevant documents. We show that our techniques are superior to the state-of-the-art work, according to several intuitive document relevance metrics, on several real benchmark datasets. We show results for two problem variants: finding queries to return in the highest positions the input documents (Docs2Queries-Self problem) and other relevant documents (Docs2Queries-Sim problem).

Keywords: Query by documents, similarity search, document search, competitor keyword, keyword discovery

1. Introduction

A common problem in Information Retrieval is that a user wants to retrieve documents similar to a given set of relevant documents. For example,

*Corresponding author

Email addresses: nle020@ucr.edu (Nhat X.T. Le), mshah008@cs.ucr.edu (Moloud Shahbazi), aalma021@ucr.edu (Abdulaziz Almaslukh), vagelis@cs.ucr.edu (Vagelis Hristidis)

a patent attorney may have a few documents provided by a client describing an invention and would like to search for patents similar to these input documents. Similarly, a scientist may search for related work in an area and may have access to a few documents related to this area. A Web user may have found a set of documents related to a topic, e.g., related to the topic of academic scandals, and may be looking for more similar documents on the Web.

Although there are powerful keyword search interfaces on top of various collections – LexisNexis for patent search [1], Pubmed [2], Google Custom Search API for the Web [3], etc. – a common limitation they have is that they do not allow the user to input a set of documents (one or thousands), but expect a relatively small number of terms as a query. Further, these *constrained search interfaces* typically charge a fee for every page of query results. For example, the LexisNexis Statistical Gateway charges \$0.30-\$0.40 per query, and Google Custom Search charges \$5 for 1,000 queries, up to 10k queries per day. Note that a common property in all these collections is that the user may not have access to the underlying collection through any way other than through the provided search APIs. Hence, to use these interfaces, one has to extract sets of important terms from the input documents, to formulate queries. These queries should ideally return other similar documents in high-rank positions of the results.

Given a set of input documents, this paper proposes effective techniques to generate queries that return other similar documents in high positions. We refer to this problem as *Docs2Queries*, shown in Figure 1.

The Docs2Queries problem has received limited attention by the community [4, 5]. The state-of-art works focus on extracting good terms from the input documents, given a basic understanding of the ranking formula, which is generally tf-idf (term frequency and inverse document frequency) based. Specifically, they select terms with high tf-idf score. This is a reasonable heuristic, but its drawback is that it ignores the language model of the collection and limits the heuristics to use only information from the input documents.

We propose a principled approach to select the best queries, which also considers the language model of the collection. Note that we assume no knowledge of the statistics of the collection. Instead, we build on existing sampling techniques to extract estimated statistics. *This sampling, which only retrieves a few thousand documents, occurs only once, before any query arrives, and hence its amortized cost is negligible.* Our key *hypothesis* is that

the best queries to return the input documents in high-rank positions will also return other similar documents in high-rank positions. That is, we focus on how to compute queries that will return our given input documents in high positions. Our experimental results confirm the validity of our hypothesis, and also the superiority of our method compared to the state-of-the-art methods. Specifically, our approach outperforms the state-of-the-art by up to 60% in Normalized Discounted Cumulative Gain (NDCG) of relevant documents and up to 9.8% in the number of returned similar documents.

More specifically, we follow a probabilistic approach, where for each candidate query, we compute the probabilistic distribution of positions of the input documents and pick the queries where the expected average position of the input documents is minimized. Our approach assumes that tf-idf is a key factor in the ranking formula, as does previous work [5]. Note that most popular IR ranking formulas – vector space cosine similarity, language model, probabilistic model – include a significant tf-idf factor [6].

A key challenge is the huge search space of candidate queries. Another challenge is that we have to estimate the positions' probability distributions based on the collection's estimated statistics, which are computed based on query-independent sampling as mentioned above. Another challenge is to account for the overlap between selected queries, so they do not return many common documents, given that each page of results has an access cost.

We study the performance of our techniques for two variants of the Docs2Queries problem: The *Docs2Queries-Self* variant finds queries to return the input documents in high positions. A natural application of this problem on the Web is the *Competitor's Keywords* problem, where we try to find queries for which a competitor's Web pages are ranked high. The *Docs2Queries-Sim* variant finds queries to return other relevant documents to the input documents. We show that the same techniques work well for both problems, which confirms our key hypothesis.

This paper makes the following contributions:

- It proposes a principled solution to the Docs2Queries-Self problem, which accounts for the statistical properties of the document collection.
- It shows how these techniques are also effective for the Docs2Queries-Sim problem.
- The proposed solutions are experimentally compared against state-of-the-art baseline techniques on several real datasets. Various sampling

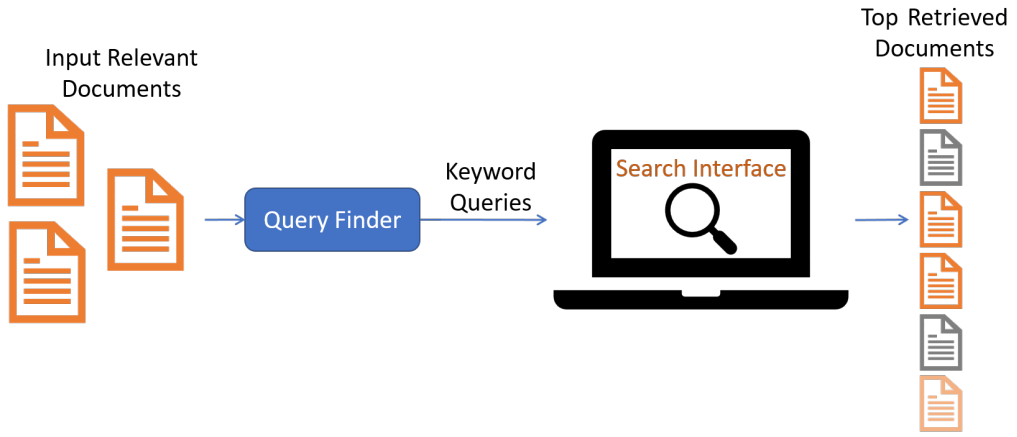


Figure 1: Docs2Queries problem overview. Our techniques power the Query Finder box. The colored returned documents are the ones that are similar to the input documents (or the input documents themselves in the Docs2Queries-Self problem variant).

techniques are used to estimate the collection statistics.

The rest of the paper is organized as follows. Related work is presented in Section 2. We define the problem variants in Section 2. The proposed algorithms are presented in Section 2. Section 5 presents the experimental evaluation, and we conclude in Section 6.

2. Related Work

In this project, the goal is to detect queries that are effective in finding similar documents to an input set of example documents using a keyword-based search interface. Here, we describe the related work on query refinement and finding similar documents.

Relevance feedback (RF): RF is a commonly used feature in information retrieval systems that uses user's interaction with the system to refine the query for improving the search results [7]. RF methods are mostly based on Rocchio's algorithm [7, 8]. In Rocchio's approach, documents and query are represented in a vector space. Every time user feedback is available, the query is refined by adding the relevant documents with a positive weight and the non-relevant documents with a negative weight.

RF is the closest line of work to our problem; however, the assumption of having the iterative improvement does not apply here because, in our case,

there is only a predefined set of documents as relevant examples and users are not generating further feedback on returned results.

Key-phrase extraction: Extracting key phrases from a document is a well-studied problem. These key-phrases could be suited to tasks like relevance filtering or browsing in retrieval [9]. Supervised approaches find key-phrases by training machine learning models for identifying key-phrases, using training documents where the key-phrases are known [10, 11, 12]. In [13], authors use dynamic query modeling to find related content based on a textual stream.

On the other hand, unsupervised methods rely on statistical information to select key-phrases [14]. A basic approach that comes to mind is to rank terms or n-grams in the input document by frequency or tf-idf score [15, 9]. Another unsupervised approach suggested in [16] involves clustering the candidate key-phrases in a document into topics, such that each topic is composed of all and only those candidate key-phrases that are related to that topic. In [5], authors select key-phrases by identifying noun-phrases using part of speech tagging. Most of the key-phrase generation methods, order the key-phrases based on a form of tf-idf score.

Search by document: Search for similar documents given an input document is a directly relevant research area to our work [4, 5]. Yang et al. [5] proposed *Query by Document* (QBD) addressing the problem of cross-referencing on-line information in the context of blogs. They select noun phrases from an input document to use as keyword queries to search for pages similar to that document. In [17], Dasdan et al. propose an approach to the covering test problem where the goal is to find out if there is a near-duplicate of a certain document in a corpus of documents that is accessible using a rate-limited keyword query interface. In both cases, the generated queries are merely based on the input document and not taking into account much of the collection statistics that matter in ranking function while computing the score of the candidate queries. Further, they rely on query generation methods to select the candidate queries.

Vidhya et al. proposed a personalized query formulation method based on identifying key phrases from an input document [18]. The key-phrases are used to query a search engine and the results are evaluated for similarity to the original document. They find the key-phrases by taking the co-occurrences within the input document. They use Jaccard similarity to measure the similarity of the retrieved document to the input document. Lee et al. [19] studied the problem of generating a search query from an

user-selected text to find similar documents. Their method relies on a supervised learning algorithm to select, rank the text's representative chunks (noun phrases, named entities), while our approach assumes no training data.

Other works on querying by documents assume full access to the whole collection, while our problem accesses the documents through a search interface. For example, Weng et al. [20] propose such indexing and ranking techniques for the collection.

Text document classification: Binary text classifiers typically input a training set of positive and negative documents and learn how to classify new documents to these two classes. Examples of those approaches are Support Vector Machine [21], Maximum Entropy [22] and recent neural network based methods such as CNN [23]. In theory, once could build such a text classifier for the input documents in our problem, and examine which keywords are used by the classifier to make a decision. That is, we can examine a trained word-based classifier to detect words that most influence the classifying of a document to be positive, and treat them as query keywords in our problem. However, a key challenge of this approach is the scarcity of the positive class, which may have only a few examples, i.e. small input document set. Moreover, it is not clear how to best combine keywords into a set of queries, for example, avoid similar words in the same query.

Document similarity functions: Document similarity is usually calculated by cosine similarity between document vectors. In recent years, neural network based embeddings such as *word2vec* [24] and *doc2vec* [25] have been shown to efficiently learn a document's distributed representation that embeds rich document semantics, especially document similarity. The requirement for these approaches to be effective is a very large collection of documents. We utilize the *doc2vec* technique to learn document vector from the entire collection to facilitate our evaluation on document similarity.

3. Definitions

We begin by defining the key data types in our problem definition and proposed algorithms. Let C be the collection of documents that are indexed by a search engine. We consider search engines that provide a keyword-based query interface to the users for accessing the documents in the indexed collection. The *keyword-based query interface* is the user interface to the Web collection that inputs a set of keywords and outputs a ranked list of documents.

A *keyword query* $Q = \{q_1, \ldots, q_n\}$ is a set of n uni-grams. As we mentioned earlier, we study the problem of finding keyword queries from a set of input documents provided by the users. Let define $I = \{d_1, \ldots, d_k\}$ as a set of k example input documents to extract keyword queries that will result in retrieving input documents at high-ranked position and/or similar documents.

Search engine providers do not reveal the exact ranking function of their system as it is an asset of the business; however, it is fair to assume that the text-based relevance ranking is a function of commonly used "term frequency inverse document frequency" (tf-idf) score. As a result, we explain our proposed solution by assuming the score of document d given keyword query Q is computed as follows:

$$S_{d,Q} = \sum\nolimits_{q \in Q} tf(d, q) \times idf(q) \tag{1}$$

where $tf(d, q)$ is the term frequency of keyword q in document d and $idf(q)$ is the inverse document frequency of q.

We present two problem variants: the ones where the goal is to return the input documents in high positions – useful for the *competitor keywords* problem – and the ones where we search for other relevant documents – useful for the *document discovery* problem. For both variants, the *budget* is the number of queries that we can submit to the search interface.

Docs2Queries-Self/Sim problem: Given a set of input documents I, a search interface, a number T of returned results per query, and maximum query length n, find a set $\mathcal{Q}$ of m queries $\mathcal{Q} = \{Q_1, Q_2, \cdots, Q_m\}$ that return as many documents R in high-ranked positions as possible. That is:

$$\mathcal{Q} = \arg\min_{\mathcal{Q}'} \frac{1}{|R|} \sum_{d \in R} \min_{Q \in \mathcal{Q}'} (pos(d, Q)) \tag{2}$$

- R is I for Doc2Queries-Self problem, and is the set of documents in C returned by query Q and similar to I for Doc2Queries-Sim problem.
- To make the problem realistic, we consider a number T as the maximum number of documents returned by a query. Documents not returned in the *top-*T are assigned position $T + 1$, that is, if $rank(d, Q)$ is the unconstrained position, then $pos(d, Q)$ is its constrained version: $pos(d, Q) = \min(T + 1, rank(d, Q))$. When a document is returned by multiple queries, its best position across all queries is considered.

There are different ways to define similarity between a document and a set of documents. We consider several similarity measures in our experiments. We also consider the simplified *Docs2Query-Self/Sim* problem, where the goal is to find a single query Q, i.e. $\mathcal{Q} = \{Q\}$. Note that the combination of T and m represents the *budget* of our problem. For example, one needs to pay \$5 per 1000 queries in Google Custom Search [3].

Solving Docs2Queries-Sim directly in a principled way is hard because R is unknown. For that, we hypothesize that queries that will return the input documents I in high-ranked positions (Docs2Queries-Self problem), will also return other similar documents. As we see in Section 5, our experiments support our hypothesis.

Note that our techniques, as well as the baselines methods we compare against, assume a tf-idf ranking formula. We understand that some search interfaces –especially Web search engines– use many more features for ranking, but tf-idf is still a significant factor.

4. Algorithms

In this section, we explain our approach to solve the Docs2Queries-Self problem. As mentioned before, this approach is also effective for the related Docs2Queries-Sim problem. In order to make the presentation simple, we start by showing how to solve the problem for a single query with a single keyword, then extend our solution to a single query with multiple keywords. Finally, we explain our approach for the case of multiple queries including multiple keywords, according to Equation 2. As mentioned earlier, we assume that the search interface's score function for document d with respect to a query Q is computed using Equation 1.

The rank position of document d within the collection given query Q is defined as follows:

$$rank(d, Q) = P(S_{d',Q} > S_{d,Q}) \times |C| = (1 - LowP_{d,Q}) \times |C| \qquad (3)$$

where P is a shorthand for *probability*, $|C|$ is the collection size, $S_{d',Q}$ is the score of a random document d' in collection with respect to Q, and $LowP_{d,Q}$ is the probability that a random document d' has score less than or equal to the score of the input document d regarding to query Q: $LowP_{d,Q} = P(S_{d',Q} \leq S_{d,Q})$ (4). In practice, only up to top T documents are returned per query, where T is specified by the search interface or by the access budget,

thus $pos(d, Q) = \min(T + 1, (1 - LowP_{d,Q}) \times |C|)$ (5). That is, we treat all the documents that are ranked lower than top T equally, and place them in $(T + 1)$th position in the ranking.

4.1. Single-keyword query

We start by describing our solution for the simplest case where the goal is to find the best query $Q = \{q\}$ with a single keyword. Given this condition, $LowP_{d,Q} = P(S_{d',Q} \leq S_{d,Q}) = \sum_{k \leq tf(d,q)} P(tf(d', q) = k)$, where $tf(d', q)$ is the term frequency of keyword q for a random document d' in collection.

From Docs2Query-Self problem's Equation 2 and Equation 5, we can now define the objective function to find a single-keyword query based on term frequency distribution as follows:

$$q = \arg\min_{q'} \sum_{d \in I} pos(d, q') = \arg\max_{q'} \sum_{d \in I} \max(1 - \frac{T+1}{|C|}, LowP_{d,q'}) \quad (6)$$

$$= \arg\max_{q'} \sum_{d \in I} \max(1 - \frac{T+1}{|C|}, \sum_{k \leq tf(d,q')} P(tf(d', q') = k)) \quad (7)$$

In Equation 6, we try to maximize the probability of random documents being being ranked below the input documents d ($LowP_{d,Q}$). $1 - \frac{T+1}{|C|}$ is the probability that a random document has the rank out of top T. When document d is in top T documents returned by query q': $\max(1 - \frac{T+1}{|C|}, LowP_{d,q'}) = LowP_{d,q'}$. In Equation 7, the only unknown terms are the collection size ($|C|$) and $P(tf(d', q) = k)$, i.e., the term frequency distribution, which we both estimate using a sampling technique over the search interface, which occurs once before any query arrives. This sampling strategy is discussed in detail in the Section 5.

In practice, to avoid having to evaluate Equation 7 for all terms q' extracted from the input documents I, we only consider the l terms with the highest tf-idf score, where tf is computed on I by viewing all documents in I as a single concatenated document, and idf on the whole collection C. In the experiments we set l=20, which in practice we found that it is equivalent to not setting a l value (i.e., the same query is returned for both $l = 20$ and no l setting).

4.2. Multi-Keyword Query

In order to compute the position of a document given a query with n keywords, we assume that term occurrences are independent within a document,

which is a common assumption in Information Retrieval ranking functions. Hence, we define the objective function to find the best query as follows:

$$Q = \arg\min_{Q'} \sum_{d \in I} pos(d, Q') = \arg\max_{Q'} \sum_{d \in I} \max\left(1 - \frac{T+1}{|C|}, LowP_{d,Q'}\right) \quad (8)$$

We expand Equation 4 for a multi-keyword query as follows:

$$LowP_{d,Q} = P(S_{d',Q} \leq S_{d,Q}) \approx \sum_{k_i | \sum_{i=1}^{n} k_i \times idf(q_i) \leq S(d,Q), k_i < z} \prod_{j=1}^{n} P(tf(d', q_j) = k_i) \quad (9)$$

Note that, we assume that query terms have a maximum term frequency z in any document of the collection to make Equation 9's computation practical. Without this, a naive approach to solve Equation 8 given Equation 9 is to consider all keyword combinations of size n for the keywords within vocabulary of input documents. Our assumption is reasonable for large values of z (we use $z = 20$ in the experiments), as a query term is not a stop word.

Similar to Equation 7, the only unknown terms in Equation 8 and 9 are collection size $|C|$ and term frequency distribution $P(tf(d', q_j) = k_i)$. We approximate the document score $S_{d,Q}$ by Equation 1, which translates our assumption that tf-idf is a key factor in the search engine's ranking formula.

We also prune the vocabulary terms to maximum l query term candidates as before. There are $\binom{l}{n}$ possible queries. Each query requires $O(z^n)$ to compute its probability score according to above equation, so that is $O(\binom{l}{n} \times z^n)$ in total. To scale up the computation, we can take the exponential of both sides to turn the right-hand side into multiplication operation, for which we can utilize a vectorized implementation with off-the-shelve numerical libraries. These exact parameters used in our experiments can be found in Table 4 in Section 5. Note that, so far we explained how to find a query with exactly n keywords. We repeat the computation for queries with one keyword up to n, and select the best query among them.

4.3. Best Position (BP) Algorithm: Multiple Multi-Keyword Queries

Finally, we are ready to present our full algorithm, which we refer as Best Position (BP) algorithm, which finds m queries with up to n keywords. A key challenge is that there is an exponential number of combinations of m queries. For this reason, we propose heuristics to finds m queries as follows.

BP-Cluster: this heuristic first partitions the input documents into m clusters using K-Nearest-Neighbor (KNN) algorithm. Then it finds the best

query per cluster. Note that as the similarity measure in KNN algorithm, we use the cosine similarity of the document vectors with tf-idf weights. Algorithm 1 shows the procedure to produce m queries.

Algorithm 1 Best Position with Cluster heuristic.

procedure BP-CLUSTER(I, m, n)
    I: Set of input documents
    m: Number of queries
    n: Maximum number of keywords per query
    for ($i = 1; i <= m; i++$) **do**
        $C_i = \varnothing$ ▷ document cluster i
        Randomly select a document from I, and add to C_i
        add $(|I|/m) - 1$ closest documents to C_i.
        Find a new, best query $Q, |Q| \leq n$ using Eq. 8 and 9
        Remove C_i from I

BP-Greedy: this variant maintains the best positions of each input document, given the current query set. Then, a query is greedily selected based if it improves the sum of best positions by a maximum amount. The details are given in Algorithm 2.

Note that even though these algorithms do not target Docs2Queries-Sim problem directly, our aforementioned hypothesis implies that these algorithms will also achieve good performance in Docs2Queries-Sim problem. We show empirical evidence supporting this in Section 5.

Algorithm 2 Best Position with Greedy heuristic

procedure BP-CLUSTER(I, m, n)
    I: Set of input documents,
    m: Number of queries,
    n: Maximum number of keywords per query
    $doc\_ranks = \varnothing$ ▷ dictionary: input doc → current rank
    for ($i = 1; i <= m; i++$) **do**
        Use Eq. 9 to find a query $Q, |Q| \leq n$ that improves $\sum doc\_ranks.values()$ the most.
        update doc_ranks with Q

5. Evaluation Results

5.1. Experimental Settings

We use Lucene search engine with our tf-idf based scoring function given by Equation 1 as the search interface in our experiments. In this section, we describe the experimental results comparing our BP algorithm to the state-of-the-art Query-by-Document (QBD) method, and one more baseline method.

Datasets: In our experiments, we use the "TREC-9 Filtering Track" test collection [26], which contains 348,566 abstracts of references from MEDLINE, which is an online medical information database, consisting of titles and/or abstracts from 270 medical journals over a five-year period (1987-1991). The available fields include title, abstract and a few other fields [27]. As the set of input documents, we use filtering topics that have at least 8 documents to get stable experimental results. These filtering topics are from the original query set developed by Hersh et al. [27] for their IR experiments. As a second dataset, we use "TREC-8 Ad hoc Test Collections" [28]. This corpus contains about 1.6 million English articles from news media such as The Wall Street Journal and Financial Times Limited and 50 natural language topics. We remove documents with less than 10 words. Table 1 describes the details of our datasets.

Table 1: Dataset characteristics.

	TREC-9	TREC-8
#Documents	233,444	1,634,243
Average Document Length	84	200
Vocabulary Size	38,849	254,102
#Query Sets	63	50
Average Size of Query Sets	30	237
Average Document Frequency	505	1,287

Evaluation measures: To evaluate the proposed methods for the *Docs2Queries-Self* problem, we use all topic documents as input to generate the queries, and then use the positions of the same set of documents to evaluate the various measures. In contrast, for *Docs2Queries-Sim*, we divide the topic documents into two halves: a *source set* and a *target set*. The source set is used to compute the queries, and the target set to evaluate the effectiveness

Table 2: Evaluation Measures and applicable problems

Measure	Description	Docs2Queries-Self	Docs2Queries-Sim
NDCG	Normalized discount cumulative gain	x	x
MAP	Mean average precision at rank T	x	x
Average Position	Average position of the relevant documents	x	x
Recall	Ratio of relevant documents that are successfully retrieved	x	x
Average Sim. - pairwise	Average similarity of returned documents with the closest input one		x
Average Sim. - centroid	Average similarity of returned documents with input set's centroid		x
Sim. Doc Count - pairwise	# similar documents returned using pairwise similarity		x
Sim. Doc Count - centroid	# similar documents returned using centroid-wise similarity		x

in discovering other relevant documents. The intuition is that target documents can be considered golden standard for similar documents of source set, thus we expect them to be ranked in high positions. When we are finding multiple queries per input set ($m > 1$), a document may be returned by some of those queries. In that case, we consider the best rank across all queries for that document.

Our first set of evaluation measures includes measures applicable for both Docs2Queries-Self/Sim problems:

- *Normalized Discount Cumulative Gain (NDCG)* with 1-0 relevance values.
- *Mean Average Precision (MAP)* of relevant document up to top T, which is set upfront (Table 4).
- *Average Position* of the relevant documents. If a relevant document is returned in multiple queries, we only record its best rank.
- *Recall* representing the ratio of relevant documents that are successfully retrieved using the found queries. Recall is in the range from 0 to 1.

Our second set of measures aims to estimate the similarity of all retrieved documents compared to the source documents, thus these measures are suitable for the Docs2Queries-Sim problem. In our evaluation, we model documents by their distributed representations using *doc2vec* technique which is

proposed by Quoc Le et al. [25] and is has been shown to effectively maintain document semantics, especially document similarity. To compute document vectors, we utilize Gensim library [29] with document vector of size 50, trained after 50 epochs. Cosine similarity is utilized to calculated similarity of two documents. We extend this to estimate similarity between a document and a document set in two flavors: 1) pairwise: similarity between the document and its most similar one in the document set; and 2) centroid-wise: similarity between the document and the document set's centroid. Based on that we employ four intuitive measures as follows:

- *Average Similarity with pairwise* similarity flavor.
- *Average Similarity with centroid-wise* similarity flavor.
- *Similar Doc Count - pairwise*: the number of similar documents in retrieved ones, where a document is considered similar if its pairwise similarity with the source set is higher than a threshold. We use the average pairwise similarity in the source set as the threshold.
- *Similar Doc Count - centroid-wise*: similar to the previous one except that we use the similarity between source set's centroid and the least similar source document as the similarity threshold.

We summarize our evaluation measures in Table 2.

Baseline techniques: We compare our methods to two baselines described below.

State of the art – Query by Document (QBD): QBD was proposed by Yang et al. to automate the process of cross referencing on-line information content [5]. Given a single document, they first extract noun-phrases (i.e. sequences of nouns and adjectives) as candidate queries using part of speech tagging tools. Then, they sort those candidates based on the following scoring function that accepts a phrase candidate P and an input document d:

$$score(P, d) = \sum_{t \in P} tf(d, t) \times idf(t) + \alpha \times coherence(P) \tag{10}$$

Where $coherence(P)$ is defined as follows:

$$coherence(P) = \frac{tf(d, P) \times (1 + \log tf(d, P))}{\frac{1}{|P|} \times \sum_{t \in P} tf(d, t)} \tag{11}$$

The authors also propose another scoring function based on co-occurrence information of the terms in a phrase; but we omit that method because it results in worse performance than the one in Equation 10. We naturally extends QBD to consider all noun phrases in the input set of multiple documents in our context. We carry out an experiment to find the optimal coherence weight α that turns out to be 1 in most cases. We include this experiment's details in the Appendix.

Tf-Idf based method: In addition to QBD, we define the following baseline (called TF-IDF) to compare against our proposed solutions for finding m queries with up to n keywords each:

1. Select top $n \times m$ keywords with the highest sum of tf-idf in the concatenated input documents.
2. Then the first n keywords are assigned to the first query, the next n keywords to the second query, and so on.

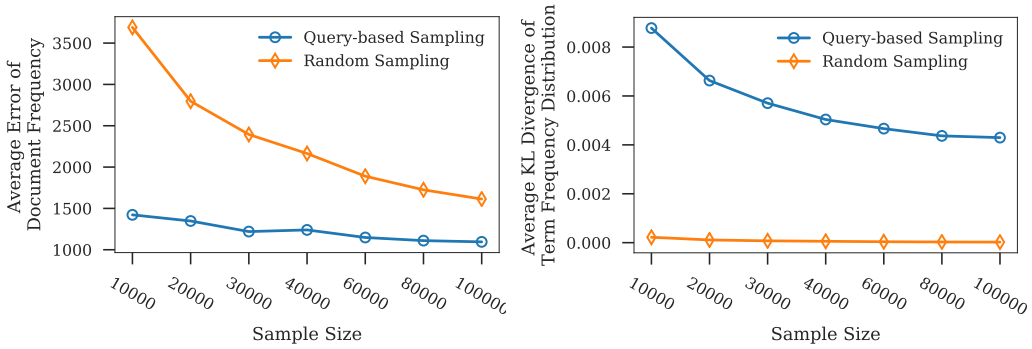
Randomized Fingerprint method: Dasdan et al. [17] propose an approach to check if there is a near-duplicate of a certain document in a corpus of documents hidden behind a query interface. The essence of this approach is to select random, long n-grams as search queries. Targeting the near-duplicate problem, this method is efficient when the query is very long and the ranking model highly consider keyword proximity. Note that, we do not implement the phase of fixing one fingerprint per document since its repeatability goal is orthogonal to the retrieval performance.

Estimating corpus statistics: All proposed algorithms and baselines require some statistics from the collection, in addition to the statistics of the input documents I. Table 3 shows different collection statistics required by each algorithm. *Interestingly, our methods use collection statistics on both tf and idf, whereas the baselines only on idf. We clarify that all methods use the tfs in the input documents I, but our methods are the only ones that compare the tf of each term to the distribution of tfs of this term in the whole collection. We argue that this is a key reason that explains why our methods perform better.* We estimate the collection term frequency distribution based on the distribution driven from a sample set.

Random Sampling: The first approach is to assume that a representative subset of the underlying collection is available. In our experiments, we generate this subset, by randomly selecting a set of sample documents from datasets that are used in our experiments.

Table 3: Collection statistics requirements. Inverse Document Frequency - idf, Term Frequency - tf.

	idf	tf distribution - $probability(X_q)$
TF-IDF	x	
QBD	x	
BP-Cluster	x	x
BP-Greedy	x	x



(a) Average error of estimated vs. actual document frequency.

(b) Average KL divergence of estimated and actual term frequency distribution.

Figure 2: Error rate of sampling strategies in TREC-8 dataset.

Query-based Sampling: We follow an approach proposed by Ipeirotis et al. [30] to compute document frequency statistics. The sampling for collecting a set of documents is an iterative process with the following steps:

Start by choosing a seed query keyword. While the number of sampled documents is less than required (sample size is a parameter), do:

1. Select a random keyword from sampled documents vocabulary.
2. Issue a query using selected keyword to search interface and add the top 3 (system parameter) documents to the sample set.

Note that both sampling methods incur a fixed, one-time cost, which is independent from the later search phase. Sampling is needed for both our methods and the baselines, as they all require some collection statistics (Table 3). The query-based sampling is generally more practical, because it does not require access to the underlying collection.

Table 4: Parameters

Parameter	Value
Number of queries (m)	1, 2, 3, 4
Maximum number of keywords per query (n)	4
Number of top retrieved document threshold (T)	1,000
Maximum term frequency threshold (z)	20
Number of pruned vocabulary terms (l)	20

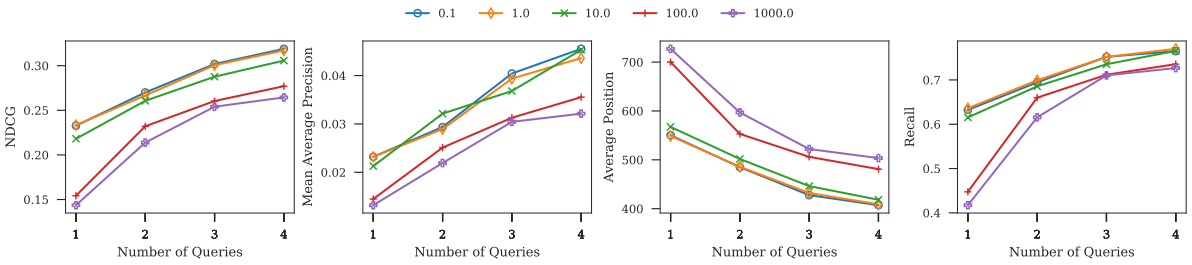
We ran both strategies 30 times. Figure 2 shows the results for TREC-8 (the results for TREC-9 were similar and were omitted to save space). Figure 2(a) shows the average absolute error of the estimated document frequencies comparing to actual document frequencies. As the size of sample set increases, we cover more words from the vocabulary of the collection, thus the absolute document frequency error drops. We further extend Query-based sampling to estimate term frequency (tf) distributions by considering sample tf distribution. We assume that the corpus size is equal to the largest estimated document frequency. Figure 2(b) indicate that our tf distribution estimation's error decreases when the sample size increases. In general query-based sampling is more accurate than random sampling in estimating term frequency distribution, but is less accurate than random sampling in estimating document frequency. Our experiments show similar, good performance results using both sampling strategies.

Finding optimal coherence weight in QBD baseline: We evaluate different values of parameter α in Equation 10 to obtain the best value to use as the coherence factor in QBD method. We demonstrate the representative results on TREC-9 dataset using two sampling techniques in Figure 3. This figure reveals that the best result is achieved by setting α to less than or equal 1. Based on this observation, for the rest of upcoming experiments, we use $\alpha = 1$ for QBD method.

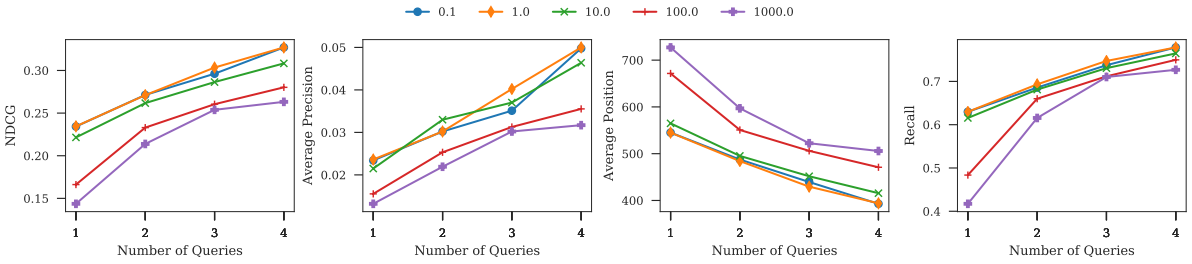
Our proposed methods' parameters: Lastly, we present all system parameters we use throughout the experiments in the next sections in Table 4.

5.2. Doc2Queries-Self Evaluation

In this section, we consider all topic documents as the input to find queries, which are then evaluated on the same document set. We vary the number of finding queries from 1 to 4 with maximum 4 keywords per query.

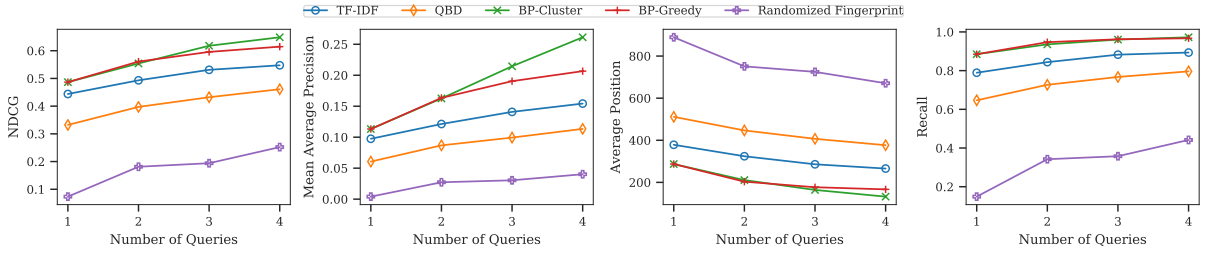


(a) Corpus statistics estimation using Query-based sampling

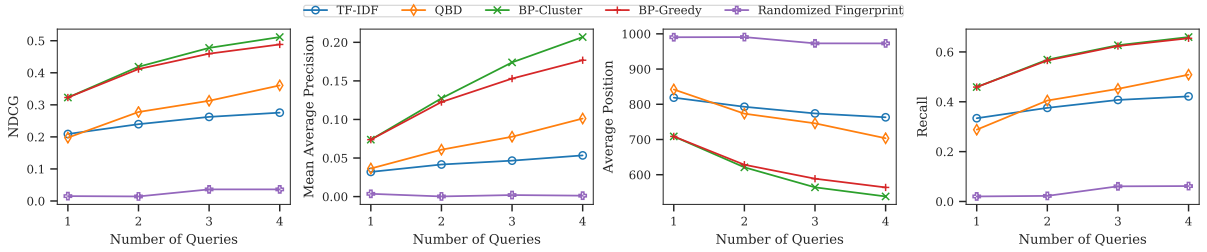


(b) Corpus statistics estimation using Random sampling

Figure 3: QBD coherence weight α on TREC-9 dataset. Higher is better for all metrics except *Average Position*.

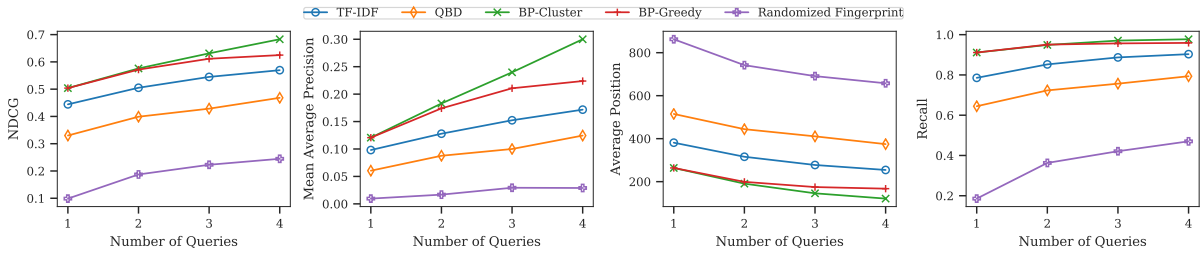


(a) TREC-9 dataset. Higher is better for all metrics except *Average Position*.

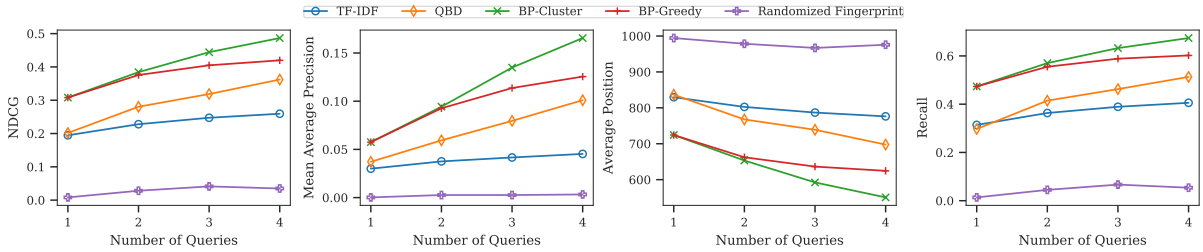


(b) TREC-8 dataset. Higher is better for all metrics except *Average Position*.

Figure 4: Performance in Doc2Queries-Self problem, using Query-based Sampling.



(a) TREC-9 dataset. Higher is better for all metrics except *Average Position*.



(b) TREC-8 dataset. Higher is better for all metrics except *Average Position*.

Figure 5: Performance in Doc2Queries-Self problem, using Random Sampling.

Figure 4 shows results on both datasets using query-based sampling strategy. In TREC-9 dataset (Figure 4(a)), our BP-Cluster method is the clear winner, followed by our BP-Greedy. We observe that BP-Cluster has Normalized Discounted Cumulative Gain (NDCG) value about 18% higher than TF-IDF baseline and about 50% higher than QBD on average. Similar or more favorable trends can be observed in other measures. Using four queries, our methods are able to return source documents at around position 150 on average, which is about 2 and 3 times better than TF-IDF and QBD respectively. Furthermore, our methods return most of source documents using four queries (*recall* ≈ 1) while the recall of TF-IDF and QBD is roughly 0.85 and 0.75 only.

In TREC-8 dataset (Figure 4(b)), our BP-Greedy and BP-Cluster methods again outperform the baselines with large margins. We notice that TF-IDF is now worsen than QBD method. Overall, all methods' performance in TREC-8 is worse than their performance in TREC-9 dataset. We believe that it is due to the fact that TREC-8 includes articles in a variety of domains and it is much larger than TREC-9. Therefore, there are fewer prominent keywords that characterize a topic set. This also explains why the TF-IDF

method is severely affected in TREC-8. In both datasets, Randomized Fingerprint method performs poorly since it picks many trivial keywords and tf-idf ranking based system does not take into account the query term order.

Note that the average position is the average rank of all source documents of a topic, thus it is expected to be higher than the average number of source documents. A topic in Trec 8 and Trec 9 dataset respectively has 236.5 and 31 documents on average, thus the average positions in these topics (Figure 4, 5) are relative to these numbers.

In Figure 5, we repeat the same experiments when the random sampling technique is used to estimate the collection statistics. The experiment's outcome for all methods does not show a noticeable change, thus we will only present experimental results using Query-based sampling in the next sections due to space limitation.

In summary, our two methods BP-Cluster/Greedy demonstrate best performances on all four measures for the Doc2Queries-Self problem. In the case of finding a single query, BP-Cluster and BP-Greedy are exactly the same. Other than that, BP-Cluster is the best algorithm.

5.3. Doc2Queries-Sim Evaluation

We report experimental results of both datasets using query-based sampling strategy in Figure 6.

Measuring target document's rank in returned set: under the first four measures in Table 2, we notice similar trends as the evaluation of Doc2Queries-Self problem (Section 5.2). Specifically, our method BP-Greedy/Cluster outperform both baselines by a large margin in all rank-based measures. This result indicates that our methods are not only able to retrieve source documents but also target documents at high positions.

The performance of all methods on target sets is lower than on source sets in rank-based measures. This is expected since all methods optimize for the source documents instead of target documents. We also notice that document positions in TREC-8 dataset are worse than in TREC-9 dataset. A reasonable explanation for this may be that the target set size in TREC-8 is larger than in TREC-9, thus target documents are retrieved in a wider range.

Measuring returned document's similarity comparing to input document set: the results using the last four measures in Table 2 are more diverse and interesting to interpret. In TREC-9 dataset (Figure 6(a)), the average similarities between retrieved documents and source documents are pretty close

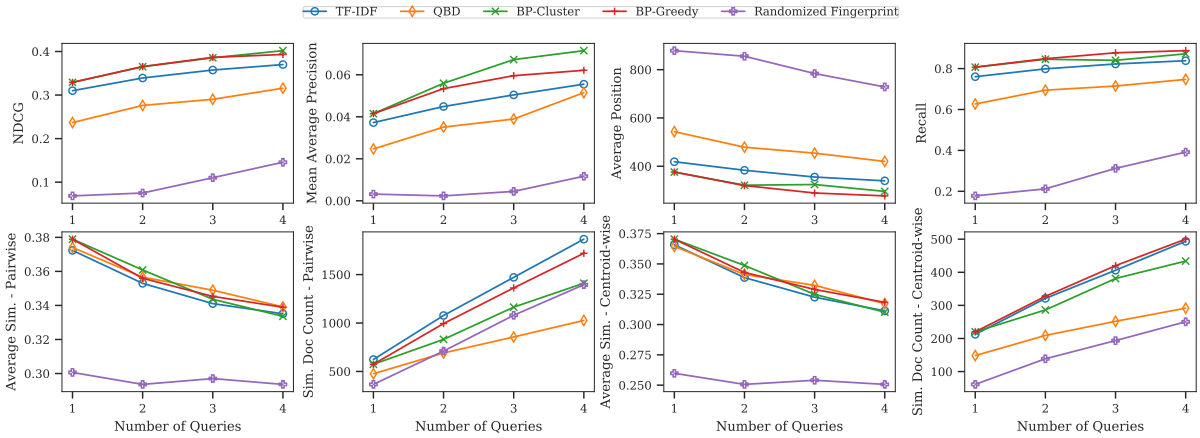
for all methods except the Randomized Fingerprint. The common trend of all methods is a decline in the average similarity when there are more queries. This is attributed to the increase of non-relevant documents at low positions when methods were asked to pick more queries. In terms of the number of similar documents (defined as having similarity larger than a threshold), TF-IDF and our BP-Cluster are the best ones. The small size and quite focusing topic nature of this dataset may have contributed to TF-IDF's success since there are common, standout words in the input documents. We will see that when these properties do not exist in a larger, more diverse dataset as TREC-8, TF-IDF based method's performance drops. Figure 6(b) reveals that our method BP-Greedy is the best one in all measures using TREC-8 dataset, followed closely by BP-Cluster. In particular, BP-Greedy has average similarity higher than QBD at most 10%, and higher than TF-IDF method at most 18.6%. In terms of retrieved document's similarity comparing to source set's centroid, BP-Greedy can achieve an average similarity 13.6% higher than QBD and 29.4% higher than TF-IDF method. Similar to previous measures, Randomized Fingerprint performs worst in most of cases. A common trend of all methods is a decline in the average similarity for more queries. This is due to the increase of non-relevant documents, given that the higher accuracy queries are generated first.

We also observe that BP-Greedy continues to improve its performance over two baselines in the measures counting for the number of relevant documents when more queries are needed. Specifically, when methods were asked for four queries, BP-Greedy is able to return about 38.9% more relevant documents than TF-IDF method, and about 19.04% more relevant documents than QBD. This result strengthen our methods' (BP-Greedy and BP-Cluster) advantage in offering not only high precision (average similarity) but also high recall (number of retrieved documents that are relevant).

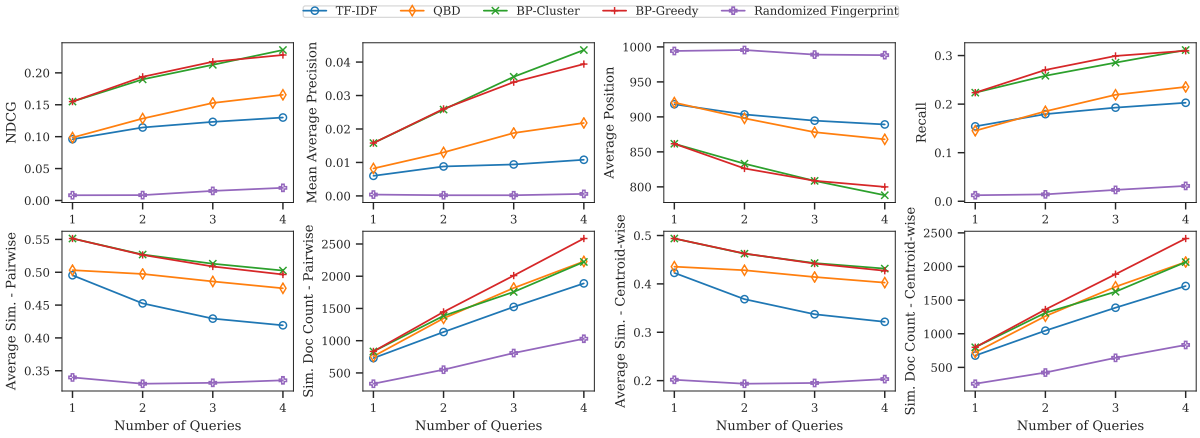
Interestingly, these results confirm our hypothesis that algorithms solving Docs2Queries-Self problem well will also achieve good performance in solving Docs2Queries-Sim problem.

6. Conclusion

We studied two variants of the *Docs2Queries* problem, and proposed a principled way to extract the best queries given a set of input documents. Our proposed solution exploits the collection's statistics in a deeper way than previous work. To estimate the collection's language model in our



(a) TREC-9 dataset. Higher is better for all metrics except *Average Position*.



(b) TREC-8 dataset. Higher is better for all metrics except *Average Position*.

Figure 6: Doc2Queries-Sim problem performance, using Query-based Sampling.

experiments, we considered two different ways of sampling the collection over a search interface. Our experimental results based on TREC-9 and TREC-8 datasets show that our BP-Greedy/Cluster algorithms outperform the state of the art baselines in both problem variants using eight different measures for both problem variants.

References

[1] R. E. Inc., Lexixnexis, https://www.lexisnexis.com/totalpatent/ (2018).

[2] U. N. L. of Medicine, Pubmed, https://www.ncbi.nlm.nih.gov/pubmed/ (2018).

[3] G. LLC, Google custom search, https://developers.google.com/custom-search/ (2018).

[4] M. D. Smucker, J. Allan, Find-similar: similarity browsing as a search tool, in: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, ACM, 2006, pp. 461–468.

[5] Y. Yang, N. Bansal, W. Dakka, P. Ipeirotis, N. Koudas, D. Papadias, Query by document, in: Proceedings of the Second ACM International Conference on Web Search and Data Mining, ACM, 2009, pp. 34–43.

[6] B. Croft, J. Lafferty, Language modeling for information retrieval, Vol. 13, Springer Science & Business Media, 2013.

[7] J. J. Rocchio, Relevance feedback in information retrieval, The Smart Retrieval System-Experiments in Automatic Document Processing (1971) 313–323.

[8] H. Chen, G. Shankaranarayanan, L. She, A. Lyer, A machine learning approach to inductive query by examples: an experiment using relevance feedback, id3, genetic algorithms, and simulated annealing, Tech. rep., DTIC Document (1998).

[9] J.-W. Lee, D.-K. Baik, A model for extracting keywords of document using term frequency and distribution, in: International Conference on

Intelligent Text Processing and Computational Linguistics, Springer, 2004, pp. 437–440.

[10] I. H. Witten, G. W. Paynter, E. Frank, C. Gutwin, C. G. Nevill-Manning, Kea: Practical automatic keyphrase extraction, in: Proceedings of the fourth ACM conference on Digital libraries, ACM, 1999, pp. 254–255.

[11] P. D. Turney, Learning algorithms for keyphrase extraction, Information retrieval 2 (4) (2000) 303–336.

[12] R. Jin, A. G. Hauptmann, Learning to select good title words: An new approach based on reverse information retrieval., in: ICML, Vol. 1, 2001, pp. 242–249.

[13] J. Foley, M. Bendersky, V. Josifovski, Learning to extract local events from the web, in: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2015, pp. 423–432.

[14] A. K. Mondal, D. K. Maji, Improved algorithms for keyword extraction and headline generation from unstructured text, First Journal publication from SIMPLE groups, CLEAR Journal.

[15] Z. Zhang, H. Cheng, Keywords extracting as text chance discovery, in: Fuzzy Systems and Knowledge Discovery, 2007. FSKD 2007. Fourth International Conference on, Vol. 2, IEEE, 2007, pp. 12–16.

[16] Z. Liu, P. Li, Y. Zheng, M. Sun, Clustering to find exemplar terms for keyphrase extraction, in: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1, Association for Computational Linguistics, 2009, pp. 257–266.

[17] A. Dasdan, P. D'Alberto, S. Kolay, C. Drome, Automatic retrieval of similar content using search engine query interface, in: Proceedings of the 18th ACM conference on Information and knowledge management, ACM, 2009, pp. 701–710.

[18] V. Govindaraju, K. Ramanathan, Similar document search and recommendation, Journal of Emerging Technologies in Web Intelligence 4 (1) (2012) 84–93.

[19] C.-J. Lee, W. B. Croft, Generating queries from user-selected text, in: Proceedings of the 4th Information Interaction in Context Symposium, ACM, 2012, pp. 100–109.

[20] L. Weng, Z. Li, R. Cai, Y. Zhang, Y. Zhou, L. T. Yang, L. Zhang, Query by document via a decomposition-based two-level retrieval approach, in: Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval, ACM, 2011, pp. 505–514.

[21] T. Joachims, Text categorization with support vector machines: Learning with many relevant features, in: European conference on machine learning, Springer, 1998, pp. 137–142.

[22] K. Nigam, J. Lafferty, A. McCallum, Using maximum entropy for text classification, in: IJCAI-99 workshop on machine learning for information filtering, Vol. 1, Stockholom, Sweden, 1999, pp. 61–67.

[23] Y. Kim, Convolutional neural networks for sentence classification, arXiv preprint arXiv:1408.5882.

[24] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: Advances in neural information processing systems, 2013, pp. 3111–3119.

[25] Q. Le, T. Mikolov, Distributed representations of sentences and documents, in: International Conference on Machine Learning, 2014, pp. 1188–1196.

[26] S. E. Robertson, D. A. Hull, The trec-9 filtering track final report., in: TREC, 2000, pp. 25–40.

[27] W. Hersh, C. Buckley, T. Leone, D. Hickam, Ohsumed: An interactive retrieval evaluation and new large test collection for research, in: SIGIR, Springer, 1994, pp. 192–201.

[28] E. M. Voorhees, D. Harman, Overview of the eighth text retrieval conference (trec-8)., in: TREC, 1999, pp. 1–24.

[29] R. Řehůřek, P. Sojka, Software Framework for Topic Modelling with Large Corpora, in: Proceedings of the LREC 2010 Workshop on New

Challenges for NLP Frameworks, ELRA, Valletta, Malta, 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.

[30] P. G. Ipeirotis, L. Gravano, Distributed search over the hidden web: Hierarchical database sampling and selection, in: Proceedings of the 28th international conference on Very Large Data Bases, VLDB Endowment, 2002, pp. 394–405.