RESEARCH ARTICLE

An index-based algorithm for fast on-line query processing of latent semantic analysis

Mingxi Zhang[1,2]*, **Pohan Li**[2], **Wei Wang**[2]

1 College of Communication and Art Design, University of Shanghai for Science and Technology, Shanghai, China, **2** School of Computer Science, Fudan University, Shanghai, China

* mingxizhang10@fudan.edu.cn

OPEN ACCESS

Citation: Zhang M, Li P, Wang W (2017) An index-based algorithm for fast on-line query processing of latent semantic analysis. PLoS ONE 12(5): e0177523. https://doi.org/10.1371/journal.pone.0177523

Editor: Quan Zou, Tianjin University, CHINA

Received: December 30, 2016

Accepted: April 29, 2017

Published: May 16, 2017

Data Availability Statement: All relevant data are within the paper, and the DBLP data are available from the DBLP website (http://dblp.uni-trier.de/).

Funding: This work was supported by Natural Science Foundation of Shanghai grant 16ZR14228, http://www.stcsm.gov.cn/; Innovation Program of Shanghai Municipal Education Commission grants 15ZZ073 and 15ZZ074, http://www.shmec.gov.cn/; and Training Project of University of Shanghai for Science and Technology grant 16HJPY-QN04, http://www.usst.edu.cn/. The funders had no role in study design, data collection and analysis,

Abstract

Latent Semantic Analysis (LSA) is widely used for finding the documents whose semantic is similar to the query of keywords. Although LSA yield promising similar results, the existing LSA algorithms involve lots of unnecessary operations in similarity computation and candidate check during on-line query processing, which is expensive in terms of time cost and cannot efficiently response the query request especially when the dataset becomes large. In this paper, we study the efficiency problem of on-line query processing for LSA towards efficiently searching the similar documents to a given query. We rewrite the similarity equation of LSA combined with an intermediate value called partial similarity that is stored in a designed index called partial index. For reducing the searching space, we give an approximate form of similarity equation, and then develop an efficient algorithm for building partial index, which skips the partial similarities lower than a given threshold θ. Based on partial index, we develop an efficient algorithm called ILSA for supporting fast on-line query processing. The given query is transformed into a pseudo document vector, and the similarities between query and candidate documents are computed by accumulating the partial similarities obtained from the index nodes corresponds to non-zero entries in the pseudo document vector. Compared to the LSA algorithm, ILSA reduces the time cost of on-line query processing by pruning the candidate documents that are not promising and skipping the operations that make little contribution to similarity scores. Extensive experiments through comparison with LSA have been done, which demonstrate the efficiency and effectiveness of our proposed algorithm.

Introduction

Many real data sets could be grouped as documents, including as web pages, literature and product profiles. With such data sets becoming massive and diverse, there is a need for designing algorithmic tools and developing applications to discover the underlying relationship from the data. Consider an example of the document search in a dataset, even though a document is on precisely the same topic to a input query of keywords, it may not be searched when its contained terms are different to the input keywords. In previous work, there are some semantic approaches that can be used finding the documents whose semantic is similar to the query of

decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

keywords, e.g., Latent Semantic Analysis (LSA) [1–4], Probabilistic LSA (PLSA) [5, 6], Latent Dirichlet Allocation (LDA) [7–10] and latent factorization model (LFM) [11, 12]. Among these approaches, LSA is a well-known representative which has been widely applied to various research fields, including document retrieval [13, 14], query expansion [15, 16], data extraction [17, 18] and text classification [19, 20]. For improving the performance of these applications, LSA provides an effective function for searching the similar documents for a given query of keywords. Specifically, LSA represents the relationship between documents and terms by a term-document matrix that is further decomposed into a product of three other matrices by the singular value decomposition (SVD) [1, 3, 4]. SVD is the mathematical tool behind LSA and some applications including association prediction [21], similarity computation [22, 23], clustering [24, 25], images analysis [26] and collaborative filtering [27, 28]. For the given query, LSA transforms it into a pseudo document vector and computes the similarities between query and candidate documents over the SVD result of the term-document matrix.

LSA has also been applied to other research fields recently, including social data analysis [29, 30], collaborative filtering [31–33], sign language translation [34] and gene sequence analysis [35–40]. For example, in the field of social data analysis, [29] adopted LSA for producing better annotated video clip in social multimedia data. [30] measured the semantic similarity in the text of social media by using a topic-based LSA. For obtaining better result of recommendation, [31] proposed a latent class regression recommender system (LCRRS) through extending PLSA for collaborative filtering based on cluster-wise linear regression. [32] presented a component recommender approach based on LSA to initialize the word distributions for different topics. [33] fused LSA and K-means for better recommendation of antiarrhythmic drugs through capturing the latent factors between the arrhythmia types and patients. In the field of sign language translation, [34] used de-bruijn graph with LSA in the decoding process to improve the quality and accuracy of translation result. For discovering the associations between genes and diseases, [35] computed the similarities between genes using LSA algorithm, and divided the similar cardiovascular disease (CVD) association genes into different clusters. [36] applied the latent factorization model (LFM) to predict the genes related to diseases by representing the relationship between genes and diseases with the gene-disease association matrix [37, 38]. [39] employed LSA to visualize gene expression experiments and defined an asymmetric similarity measure of association for genes by using the correspondence word-gene document-experiment. [40] presented a text mining approach based on LSA for prioritization, clustering and functional annotation of miRNAs.

Nevertheless, although LSA yields promising similar results and provides an effective solution to above applications, however, lots of unnecessary operations are involved in similarity computation and candidate check during on-line query processing, which make it expensive in terms of time cost and cannot efficiently response the query request especially when the dataset grows large.

Some optimization techniques on LSA have been developed recently. [41] proposed a faster optimization algorithm for solving the Non-negative Sparse Latent Semantic Analysis (NN-Sparse LSA), and implemented the parallel version of the fast NN-Sparse LSA algorithm by parallel programming framework of the Compute Unified Device Architecture (CUDA). [42] proposed an on-line belief propagation for PLSA to handle big data streams by splitting the data stream into a set of small segments and uses the estimated parameters of previous segments to calculate the gradient descent of the current segment. [43] proposed a randomized SVD algorithm that scales the original matrix to a small matrix by sampling a constant number of rows or columns of the matrix, and the SVD of the original matrix is derived approximately by computing the SVD of the small matrix. [44] proposed an incremental SVD algorithm, which can update the SVD of a given matrix dynamically by adding rows and columns of data without re-computing the SVD result. [45] proposed an algorithm for incrementally computing the left singular vectors of SVD result by exploiting the relationship between the *QR*-decomposition and the SVD. QUIC-SVD [46] provides an algorithm which producing the approximation of the whole-matrix SVD based on a sampling mechanism called the cosine tree, and provides speedups of several orders of magnitude over exact SVD. [47, 48] proposed an algorithm for accurately computation of SVD by inhering the high accuracy properties of the Jacobi algorithm [49]. [50] introduced a bi-iteration type subspace tracker for updating SVD approximation of the cross-correlation matrix of dimension $N \times M$. [51] designed a secure, correct, and efficient protocols for outsourcing the SVD of a malicious cloud. [52] proposed an algorithm for extremely fast dimensionality reduction by employing the Gaussian-based random projection and a Hadamard-based random projection. However, the above approaches mainly focus on improving the efficiency in the pre-computation stage, few of them pay attention to the efficiency problem of on-line query processing.

In this paper, we study the efficiency problem of on-line query processing for LSA, towards efficiently searching the similar documents in large dataset. We rewrite the similarity equation of LSA combined with an intermediate value called partial similarity, and divide the similarity computation into two steps: the first step is to compute the partial similarities, and the second step is to compute the similarities between query and candidate documents based on the partial similarities. The partial similarities are computed in the off-line stage and stored in a designed index called partial index. For reducing the searching space during query processing, we give an approximate form of similarity equation, and then develop an efficient algorithm for building partial index, which skips the partial similarities lower than a given threshold θ. The similarities between query and candidate document is computed in the on-line stage, and an efficient algorithm called ILSA is developed for supporting fast on-line query processing through searching similar documents from the partial index. For a given query of keywords, we first transform it into a pseudo document vector and then compute the similarities between query and candidate documents by accumulating the partial similarities obtained from the partial index. ILSA accesses only the partial index nodes corresponds to non-zero entries in the pseudo document vector, which prunes candidate documents that are not promising and reduces the unnecessary operations on similarity computation that make little contribution to similarity scores. By extensive mathematical analysis, we give the maximal upper bound of the difference between ILSA and naive LSA under threshold θ. Extensive experiments through comparison with LSA have been done, which demonstrate the efficiency and effectiveness of our proposed algorithm.

Methods

Preliminaries

Before we discuss further on LSA, we first list the definition of correlation matrix of term-document for the subsequent discussions.

Definition 1 (Correlation Matrix of Term-Document). *A Correlation Matrix of Term-Document is formalized as a $M \times N$ matrix $C_{M \times N}$, where M is size of term set T and N is the size of document set D. In which, the entry C_{t_i,d_j} represents the correlation between term t_i and document d_j, which is initialized as the number of times that term t_i occurs in document d_j.*

LSA maps each document into a M-dimension vector and forms a correlation matrix of term-document C. Unlike precise matching method, the matrix C is decomposed by SVD, that compresses matrix C into a new low-dimension space to remove the noise terms. SVD can not only reduce the scale of the data, but also find the underlying relationship between terms. During the on-line query processing, the input terms are firstly transform into a query vector of pseudo document, and then LSA uses cosine coefficient to compute the similarity between query vector and the low-dimension vector corresponds to each document over the decomposition result of matrix C. The candidate documents are sorted according to the corresponding similarities, and then returned to current user. Besides cosine, other measure can also be used for computing similarity, such as Jaccard coefficient and dot product, and without loss of generality we choose cosine to measure the similarity. Specifically, the procedure of LSA can be summarized as follows.

1. Building term-document correlation matrix C by analyzing document set D. For each document $d_i \in D$, transform it into vector form $V_i(v_1, v_2, \ldots, v_M)$, where v_j refers to C_{t_i,d_j} as described in Definition 1, that is computed by counting the number of times that term t_j occurs in document d_i. Precisely, v_j is usually defined by the normalized TF*IDF (term frequency inverse * document frequency) model [53, 54] that is widely used for measuring the term weights in a document set [55, 56]. Specifically, the entry C_{t_i,d_j} is assigned as the TF*IDF of term t_i that occurs in document d_j. After normalizing vector V_i for each document $d_i \in D$, the term-document correlation matrix C is represented as:
$$C = (V_1, V_2, \ldots, V_N) \tag{1}$$

2. Singular value decomposition (SVD) of term-document correlation matrix C. For a term-document correlation matrix C, there exists a decomposition such that
$$C = USV^T \tag{2}$$
where U is an $M \times M$ matrix, the column of U is the orthogonal vector of matrix CC^T, and C^T is the transpose of C; S is an $M \times N$ matrix, $S_{i,i} = \sqrt{\lambda_i}$, and λ_i is the i-th biggest eigenvalue of CC^T; and V is an $N \times N$ matrix, the vector of V is the orthogonal vector of matrix $C^T C$, and V^T is the transpose of V.

3. Get low rank approximation matrix of matrix C. The r-dimension rank approximation matrix of C can be described as:
$$C_r = U_r S_r V_r^T \tag{3}$$
where U_r and V_r are calculated by discarding the columns of U and V from $r + 1$ on, S_r are calculated by discarding both columns and rows from $r + 1$ on, and $r \ll M$. The noisy terms can be removed by setting r, but some informative terms would be ignored when r is set too small. On the other hand, when r is set too big, some noisy terms would be involved.

4. On-line query processing for input keywords. Given a query Q of keywords, the procedure of on-line query processing is described as follows. First, view this as a vector of a mini document and transform it into a pseudo document vector $\hat{Q}$ of low-dimensional space according to the result of SVD, described as:
$$\hat{Q} = S_r^{-1} U_r^T Q \tag{4}$$
where S_r^{-1} is the inverse matrix of S_r. Second, compute the similarity between Q and document $d_i \in D$ by the cosine value between $\hat{Q}$ and the column vector $V^T(:, i)$, described as:
$$\mathrm{sim}(Q, d_i) = \mathrm{cosine}(\hat{Q}, V_r^T(:, d_i)) = \frac{\sum_{t_j} \hat{Q}(t_j) V_r^T(t_j, d_i)}{\sqrt{\sum_{t_j} (\hat{Q}(t_j))^2} \sqrt{\sum_{t_j} (V_r^T(t_j, d_i))^2}} \tag{5}$$

And finally, find top k most similar documents ranking from the document set such that $\mathrm{sim}(Q, d_i) \geq \mathrm{sim}(Q, d_x)$ for d_i in the returning list and d_x not, and then sort them with similarities descending in the returning list.

Rewrite LSA similarity equation

During on-line query processing of LSA, two factors that increase the computational cost are involved. First, the more candidates to check, the more time the algorithm will take; and second, when computing the similarity between the query and each candidate, the more terms related to the candidate, the more time will take. Therefore, the intuition to speed up the search is to prune the candidates that are not promising and reduce the unnecessary operations that make little contribution to similarity scores.

For optimizing the on-line query processing, we next rewrite the similarity equation of LSA equivalently based on Eq (5), described as:

$$\mathrm{sim}(Q, d_i) = \frac{\sum_{t_j} \hat{Q}(t_j)\mathrm{PartialSim}(d_i, t_j)}{\sqrt{\sum_{t_j} (\hat{Q}(t_j))^2}} \tag{6}$$

where $\mathrm{PartialSim}(d_i, t_j)$ is defined as:

$$\mathrm{PartialSim}(d_i, t_j) = \frac{V_r^T(t_j, d_i)}{\sqrt{\sum_{t_j} (V_r^T(t_j, d_i))^2}} \tag{7}$$

which is called the partial similarity between document d_i and term t_j. Based on this equation, the LSA similarity computation can be divided into two steps: the first step is to compute the partial similarities between documents and terms, and second step is to compute the similarity scores based on the partial similarities.

Partial index

We next introduce an index, called partial index, for reducing the searching space of LSA. The partial index used for storing the partial similarity scores in order to reduce the candidate size and optimize similarity computation. The spiritual of the partial index is similar to the pruning index proposed in our previous work in [57, 58]. An example of partial index is shown as Fig 1, where TermID denotes the term ID, DocID denotes document ID, PartialSim denotes the partial similarity, and the two-tuple ⟨DocID,PartialSim⟩ describes that the partial similarity between a document DocID and a term TermID that the document DocID belongs to is PartialSim. For example, in the set of "3276", the ⟨7181, 0.003⟩ describes that the partial similarity between document "7181" and term "3276" is 0.003, and in the set of "7801", the ⟨3058, 0.013⟩ describes that the partial similarity between document "3058" and term "7801" is 0.013.

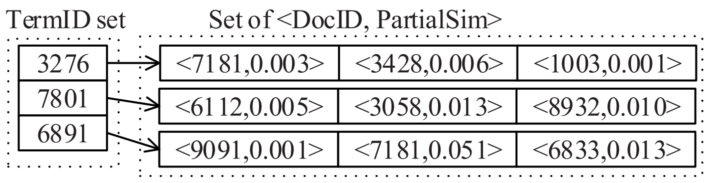

Fig 1. Example of partial index.

https://doi.org/10.1371/journal.pone.0177523.g001

Formally, the partial index is represented by a set $I = \cup_{i=1}^{|T|}\{I(t_j)\}$, where $I(t_j) = \{\langle d_i, \text{PartialSim}(d_i, t_j)\rangle | d_i \in D \wedge \langle d_i, \text{PartialSim}(d_i, t_j)\rangle \neq 0\}$. In which, $\langle d_i, \text{PartialSim}(d_i, t_j)\rangle$ is a node of the partial index corresponds to the 2-tuple of ⟨DocID, PartialSim⟩ form. Specifically, d_i is the document corresponds to DocID and PartialSim(d_i, t_j) is the partial similarity between document d_i and term t_j corresponds to PartialSim.

Approximate form of partial index

In fact, not all the terms are informative to represent the documents. For example, "SimRank: A Measure of Structural-Context Similarity" is a paper on the topic of structural-based similarity measure, so it is usually high relevant to the terms "SimRank","link", "LinkClus", "similarity" and etc., and low or not relevant to terms "phisical", "astronomy" and etc. During on-line query processing, the lower or not relevant terms would decrease the on-line query processing efficiency and even affect the quality of returned rankings.

For removing the items corresponds to terms of lower informative involved in candidate check and similarity computation, we give an approximate form of ILSA similarity equation, defined as:

$$\text{sim}_\theta(Q, d_i) = \frac{\sum_{t_j} \hat{Q}(t_j)\text{PartialSim}_\theta(d_i, t_j)}{\sqrt{\sum_{t_j}(\hat{Q}(t_j))^2}} \tag{8}$$

where PartialSim$_\theta$(d_i, t_j) is the partial similarity under threshold θ between document d_i and term t_j, defined as:

$$\text{PartialSim}_\theta(d_i, t_j) = \frac{V_r^T(t_j, d_i)}{\sqrt{\sum_{t_j}(V_r^T(t_j, d_i))^2}} \tag{9}$$

if right-hand $\geq \theta$, PartialSim$_\theta$(d_i, t_j) = 0 for otherwise.

Under the threshold θ, we next consider removing the items corresponds to terms of lower informative from the partial index. Specifically, for a 2-tuple $\langle d_i$,PartialSim$(d_i, t_j)\rangle$ in the corresponding partial index, we remove it from the partial index if the partial similarity PartialSim$_\theta$(d_i, t_j) is lower than θ. The partial index under threshold θ is denoted by a set $I_\theta = \cup_{i=1}^{|T|}\{I_\theta(t_j)\}$, where $I_\theta(t_j) = \{\langle d_i, \text{PartialSim}_\theta(d_i, t_j)\rangle | d_i \in D \wedge \text{PartialSim}_\theta(d_i, t_j) \neq \theta\}$, i.e., only the 2-tuples of non-zero partial similarities are contained in $I_\theta(t_j)$. In which, $\langle d_i$,PartialSim$_\theta(d_i, t_j)\rangle$ is a node of the partial index under threshold θ corresponds to the 2-tuple of ⟨DocID,PartialSim⟩ form, specifically, d_i is a document corresponds to DocID and PartialSim$_\theta$(d_i, t_j) is the partial similarity under threshold θ between document d_i and term t_j corresponds to PartialSim.

Fig 2 shows a partial index obtained from Fig 1 by setting threshold θ = 0.005. From this figure, we find that the index size is reduced after removing the 2-tuples ⟨7181, 0.003⟩, ⟨1003, 0.001⟩ and ⟨9091, 0.001⟩ correspond to the partial similarities lower than 0.005.

Index building algorithm

The procedure for building partial index is shown in Algorithm 1. The input of this algorithm is matrix V_r, document D and threshold θ, and the output is the partial index I_θ. In the initialization step, the partial index I_θ is set as $\emptyset$. For each term $t_j \in \{t_j || V_r^T(t_j :)| \neq 0\}$, we create node $I_\theta(t_j)$ initialized as $\emptyset$ in the partial index I_θ. And for each document $d_i \in D$, we compute PartialSim$_\theta$(d_i, t_j) and create node $\langle d_i$,PartialSim$_\theta(d_i, t_j)\rangle$ in $I_\theta(d_i)$ if PartialSim$_\theta(d_i, t_j) \geq \theta$.

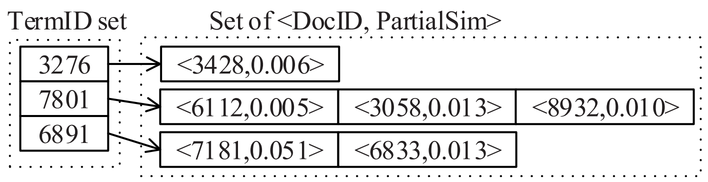

Fig 2. Example of partial index under $\theta = 0.005$.

https://doi.org/10.1371/journal.pone.0177523.g002

Algorithm 1 Algorithm for building partial index.

```
Input:
  Matrix $V_r$, document set $D$, threshold $\theta$;
Output:
  Partial index $I_\theta$;
1: Initialize $I_\theta$ as $\emptyset$;
2: for $t_j \in \{t_j || V_r^T(t_j :)| \neq 0\}$ do
3:   $I_\theta(t_j) \leftarrow \emptyset$;
4:   $I_\theta \leftarrow I_\theta \cup \{I_\theta(t_j)\}$;
5:   for $d_i \in D$ do
6:     $\text{PartialSim}_\theta(d_i, t_j) \leftarrow \frac{V_r^T(j,i)}{\sqrt{\sum_j (V_r^T(j,i))^2}}$;
7:     if $\text{PartialSim}_\theta(d_i, t_j) \geq \theta$ then
8:       $I_\theta(t_j) \leftarrow I_\theta(t_j) \cup \{\langle d_i, \text{PartialSim}_\theta(d_i, t_j)\rangle\}$;
9:     end if
10:   end for
11: end for
12: return $I_\theta$;
```

Next we analyze the time complexity of this algorithm. In the initialization stage, the time cost for creating an empty set I_θ is derived as $O(1)$. For each term t_j, the time cost for computing partial similarities between d_i and t_j for all $d_i \in D$ is derived as $O(N)$. Since only the partial similarities bigger than θ are considering for creating index nodes, so the total time cost for creating $\langle d_i, \text{PartialSim}_\theta(d_i, t_j)\rangle$ in $I_\theta(t_j)$ for all $d_i \in D$ is derived as $O(\varepsilon_{t_j} N)$, where ε_{t_j} is ratio of the partial similarities lower than θ between term t_j and all document $d_i \in D$. And then the total time cost for computing partial similarities and creating index nodes is derived as $O((1 + \epsilon)N)$. And finally, the time cost of this algorithm is derived as $O(1 + (1 + \epsilon)rN)$, where ϵ is average ε_{t_i} for all $t_j \in \{t_j || V_r^T(t_j :)| \neq 0\}$. The time cost of this algorithm is determined by the size of matrix V_r^T and threshold θ. Usually, a higher threshold θ would reduce the searching space of ILSA, and subsequently lead to lower time cost of on-line query processing, since the partial similarities corresponds to lower partial similarities are skipped when building partial index.

Index-based LSA (ILSA)

The on-line query processing procedure of the Index-based LSA (ILSA) is shown in Algorithm 2. For a given query Q, we transform it into a pseudo document vector $\hat{Q}$ and initialize $\langle \mathcal{C}, \mathcal{S}\rangle$ by setting both $\mathcal{C}$ and $\mathcal{S}$ as $\emptyset$, where $\mathcal{C}$ is the set candidate documents, $\mathcal{S}$ is the set of similarities between query and candidates, and the element $\mathcal{S}(d_i)$ in $\mathcal{S}$ is the similarity between query Q and document d_i. And then, we search the candidate documents and compute the similarities between query and candidate documents by accumulating the partial similarities obtained from the partial index nodes corresponds to non-zero entries in $\hat{Q}$. Specifically, for each term $t_j \in \{t_j | \hat{Q}(t_j) \neq 0\}$, we get each document $\langle d_i, \text{PartialSim}_\theta(d_i, t_j)\rangle \in I_\theta(t_j)$, obtain partial similarity PartialSim$_\theta$(d_i, t_j) from $\langle d_i$,PartialSim$_\theta$(d_i, t_j)$\rangle$, and then update the similarity between Q and d_i by accumulating $\frac{\hat{Q}(j)\text{PartialSim}_\theta(d_i,t_j)}{\sqrt{\sum_j(\hat{Q}(j))^2}}$. GetSortedCenter($k$, Q) is the function used for obtaining the k most similar documents according to $\langle\mathcal{C},\mathcal{S}\rangle$, the basic process of which is that firstly get the k most similar nodes from $\mathcal{C}$ according to their corresponding similarities in $\mathcal{S}$, then sort and return them.

Algorithm 2 ILSA algorithm.

```
Input:
  Matrix $U$, $X = S_r^{-1}U_r^T$, index $I_\theta$, query $Q$ and parameter $k$;
Output:
  Top-$r$ most similar sorted documents;
1: Initialize $\langle\mathcal{C},\mathcal{S}\rangle$ by setting $\mathcal{C}$ and $\mathcal{S}$ as $\emptyset$;
2: $\hat{Q} \leftarrow XQ$;
3: For $t_j \in \{t_j|\hat{Q}(t_j) \neq 0\}$ do
4:   For $\langle d_i$, PartialSim$_\theta$($d_i$, $t_j$)$\rangle \in I_\theta(t_j)$ do
5:     obtain PartialSim$_\theta$($d_i$, $t_j$) from $\langle d_i$, PartialSim$_\theta$($d_i$, $t_j$)$\rangle$;
6:     if $d_i \in \mathcal{C}$ then
7:       $\mathcal{S}(d_i) \leftarrow \mathcal{S}(d_i) + \frac{\hat{Q}(j)\text{PartialSim}_\theta(d_i,t_j)}{\sqrt{\sum_j(\hat{Q}(j))^2}}$;
8:     else
9:       $\mathcal{S}(d_i) \leftarrow \frac{\hat{Q}(j)\text{PartialSim}_\theta(d_i,t_j)}{\sqrt{\sum_j(\hat{Q}(j))^2}}$;
10:       $\mathcal{C} \leftarrow \mathcal{C} \cup \{d_i\}$;
11:       $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathcal{S}(d_i)\}$;
12:     end if
13:   end for
14: end for
15: return GetSortedCenter($k$, $\langle\mathcal{C},\mathcal{S}\rangle$);
```

The time cost of this algorithm is affected by the following three aspects. First is the time cost for transforming the given query into pseudo document, derived as $O(rN)$. Second is the time cost for choosing top k most similar documents, derived as $O(\sum_{t_j\in\{t_j|\hat{Q}(t_j)\neq 0\}}|I_\theta(t_j)| + k|\mathcal{C}|)$. And third is the time cost for sorting these k documents, denoted by $O(\Gamma(k))$, that is depends on the sort algorithm and we use selection sort in our research. So the total time cost of this algorithm is derived as $O(rN + \sum_{t_j\in\{t_j|\hat{Q}(t_j)\neq 0\}}|I_\theta(t_j)| + k|\mathcal{C}| + \Gamma(k))$.

In ILSA algorithm, we first get the non-zero entries from vector $\hat{Q}$, and then check the candidates in the partial index corresponds to the non-zero entries. Therefore, the candidate set is derived as $\mathcal{C} = \cup_{t_j\in\{t_j|\hat{Q}(t_j)\neq 0\}}\mathcal{C}(t_j)$, where $\mathcal{C}(t_j)$ is the sub candidate set corresponds to term t_j. We access only the 2-tuple $\langle d_i$,PartialSim$_\theta$(d_i, t_j)$\rangle \in I_\theta(t_j)$ in partial index I_θ during on-line query processing, so the sub candidate set $\mathcal{C}(t_j)$ is derived as $\mathcal{C}(t_j) = \{d_i|\langle d_i, \text{PartialSim}_\theta(d_i, t_j)\rangle \in I_\theta(t_j)\}$, and subsequently the candidate set $\mathcal{C}$ is derived as $\mathcal{C} = \cup_{t_j\in\{t_j|\hat{Q}(t_j)\neq 0\}}\{d_i|\langle d_i, \text{PartialSim}_\theta(d_i, t_j)\rangle \in I_\theta(t_j)\}$. When giving a higher threshold θ, the accumulation operations for computing similarities would be reduced, which consequently reduces the time cost. In this case, the size of $\mathcal{C}(t_j)$ would become smaller, and hence the size of $\mathcal{C}$ would have a downward trend. So the time cost for choosing the r centers from $\mathcal{C}$ would become lower as well. Note that the size of $\mathcal{C}(t_j)$ is equal to the size of $I_\theta(t_j)$.

Lemma 1 *For given document $d_i \in D$, term $t_i \in T$ and threshold θ, we have* $0 \leq$ PartialSim(d_i, t_j) – PartialSim$_\theta$(d_i, t_j) $\leq \theta$.

Proof. By Eqs (7) and (9), we have PartialSim(d_i, t_j) = PartialSim$_\theta$(d_i, t_j) when PartialSim(d_i, t_j) $> \theta$, which gives PartialSim(d_i, t_j) – PartialSim$_\theta$(d_i, t_j) = 0; and when

PartialSim(d_i, t_j) $\leq \theta$, we have PartialSim$_\theta$(d_i, t_j) = 0, which gives PartialSim(d_i, t_j) − PartialSim$_\theta$(d_i, t_j) = PartialSim(d_i, t_j) $\leq \theta$.

Theorem 1 *For given query Q, document $d_i \in D$ and threshold θ, we have* $0 \leq \mathrm{sim}(Q, d_i) - \mathrm{sim}_\theta(Q, d_i) \leq \theta$.

Proof. For given query Q, document $d_i \in D$ and threshold θ, by Eqs (6) and (8), we have

$$\begin{aligned}\mathrm{sim}(Q,d_i)-\mathrm{sim}_\theta(Q,d_i) &= \frac{\sum_{t_j}\hat{Q}(t_j)\mathrm{PartialSim}(d_i,t_j)}{\sqrt{\sum_{t_j}(\hat{Q}(t_j))^2}}-\frac{\sum_{t_j}\hat{Q}(t_j)\mathrm{PartialSim}_\theta(d_i,t_j)}{\sqrt{\sum_{j}(\hat{Q}(j))^2}}\\ &= \frac{\sum_{t_j}\hat{Q}(t_j)}{\sqrt{\sum_{t_j}(\hat{Q}(j))^2}}(\mathrm{PartialSim}(d_i,t_j)-\mathrm{PartialSim}_\theta(d_i,t_j))\end{aligned}$$

By Lemma 1, we have $0 \leq$ PartialSim(d_i, t_j) − PartialSim$_\theta$(d_i, t_j) $\leq \theta$, so sim(Q, d_i) − sim$_\theta$(Q, d_i) ≥ 0 and

$$\mathrm{sim}(Q,d_i)-\mathrm{sim}_\theta(Q,d_i) \leq \frac{\sum_{t_j}\hat{Q}(t_j)}{\sqrt{\sum_{t_j}(\hat{Q}(t_j))^2}}\cdot\theta = \frac{\sum_{t_j}\hat{Q}(t_j)\cdot\theta}{\sqrt{\sum_{t_j}(\hat{Q}(t_j))^2}\sqrt{\theta^2}}\cdot\theta\leq\theta$$

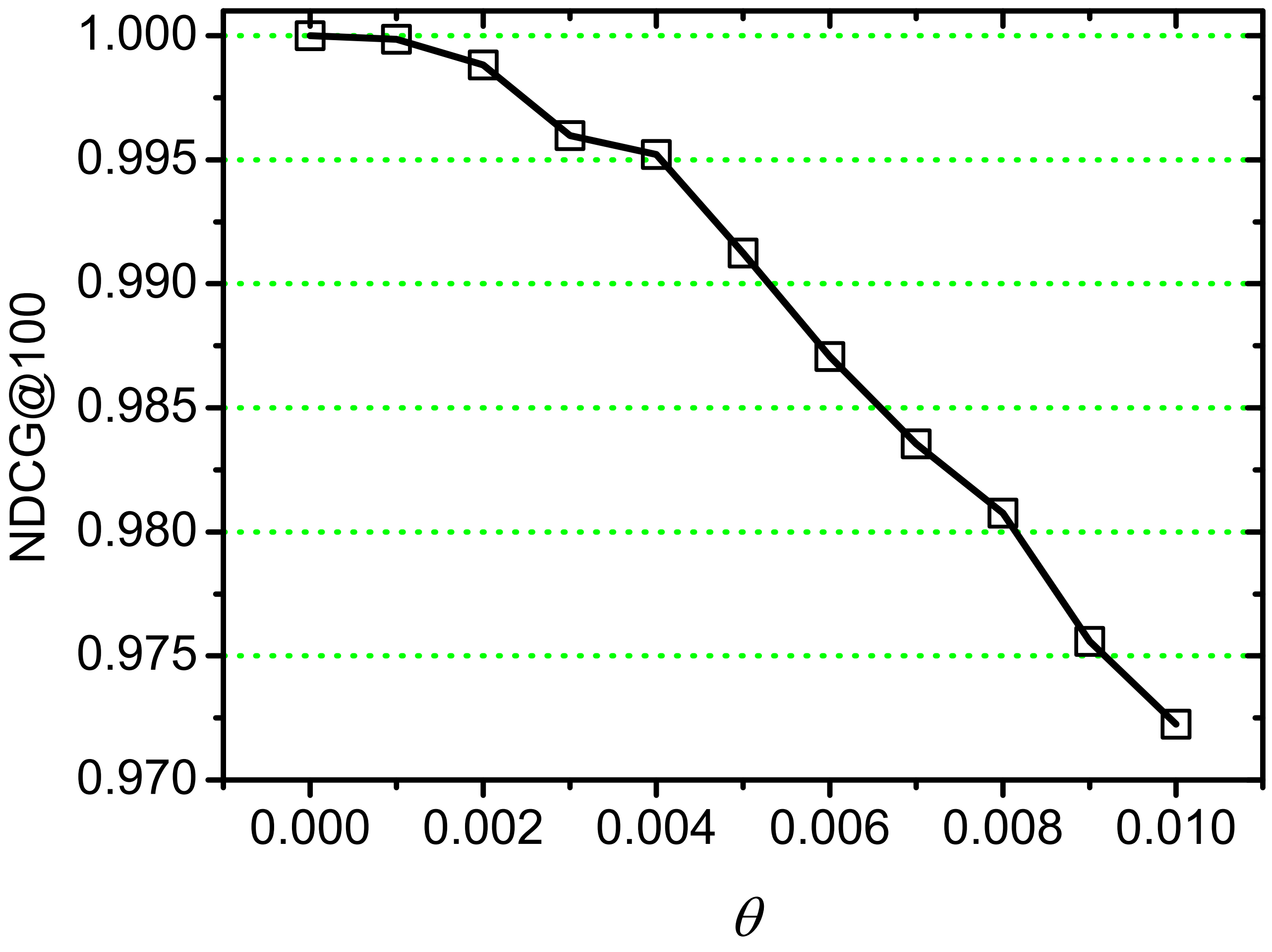

Fig 3. NDCG on varying θ.

https://doi.org/10.1371/journal.pone.0177523.g003

Theorem 1 gives the maximal difference of the maximal upper bound between LSA and ILSA, which is under control by tuning threshold θ.

Results

In this section, some preliminary experimental results are reported in real datasets. Experiments were done on a 2.90 GHz Intel(R) Core i7-3520M CPU with 8 GB main memory, running Windows 7 SP1. All algorithms were implemented in C++ and compiled by using Visual Studio C++. Net 2010.

Datasets and evaluation

The dataset used in our experiments is the set of the papers that are selected from DBLP (http://dblp.uni-trier.de/). We only keep entries of the snapshot that correspond to the papers published before March 10th, 2013. The titles of the papers that are published in SIGMOD, VLDB, SIGIR, CIKM, ICDE and EDBT conferences from 2004 to 2013 are selected. From this dataset, we choose the titles of 8,884 papers to test our algorithm and the comparisons, which contains 8,572 terms after removing the stop words, and the values of entries in term-document matrix is assigned by the TF*IDF model [53, 54].

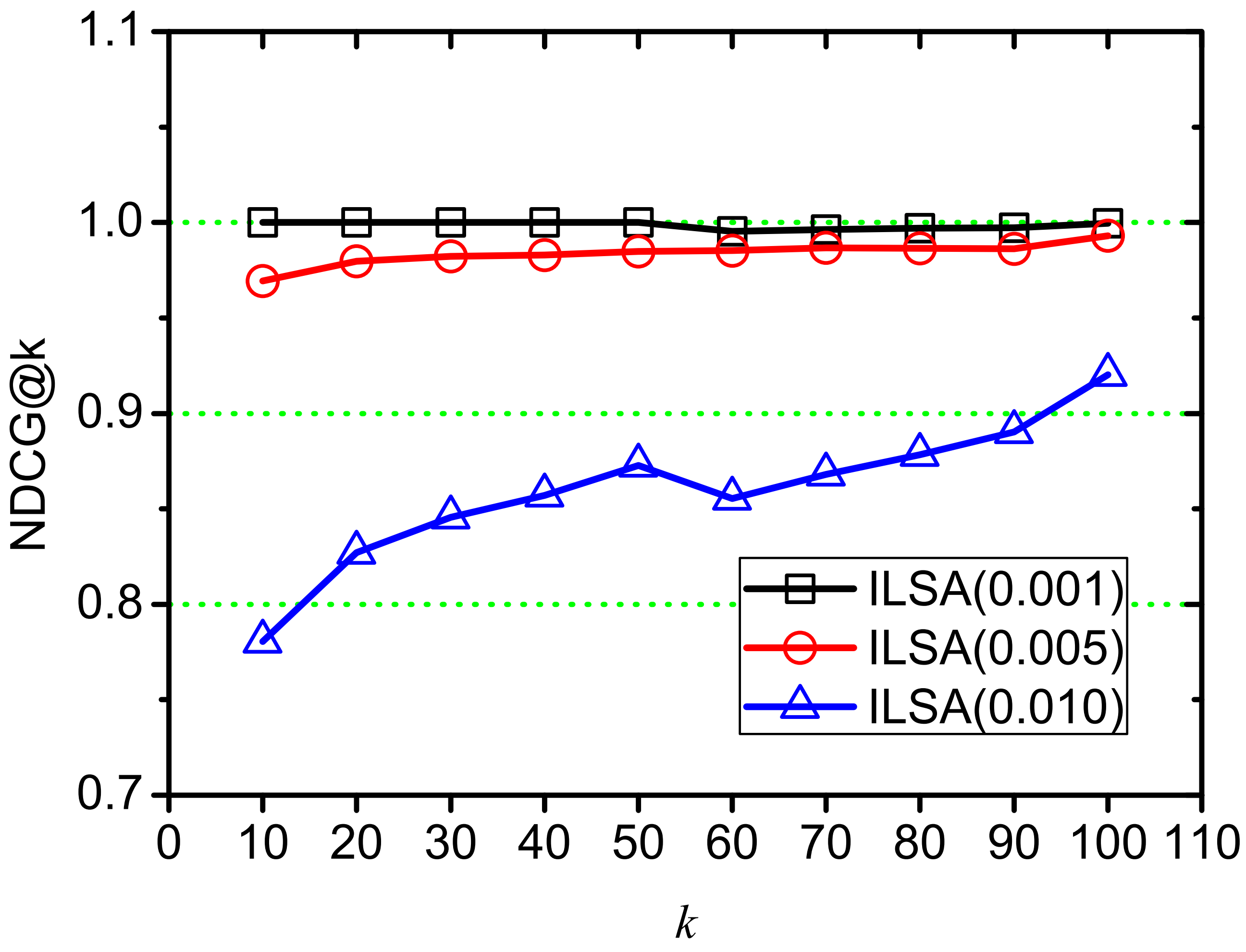

Fig 4. NDCG on varying *k*.

https://doi.org/10.1371/journal.pone.0177523.g004

We use the NDCG (Normalized Discounted Cumulative Gain) [59] to evaluate the effectiveness of returned ranking list. The NDCG@*k* (NDCG value at the *k*-th position) of the ranking result is computed by the exact LSA scores. Formally, NDCG@*k* is defined as:

$$\text{NDCG@}k = \frac{\text{DCG@}k}{\text{IDCG@}k} \tag{10}$$

where DCG@k (Discounted Cumulative Gain at *k*) is defined as:

$$\text{DCG@}k = \begin{cases} \text{REL}(v, v_i), \text{if } i < 2 \\ \text{DCG@}i + \sum_{i=2}^{k} \frac{\text{REL}(v, v_i)}{\log_2 i}, \text{if } i \geq 2 \end{cases} \tag{11}$$

where i denotes position of v_i in the returned list, REL(v, v_i) denotes the similarity score of the naive LSA between v and v_i.

Efficiency comparison includes the running time for building index, execution time of on-line query processing. In [60], extensive experiments are done in large datasets to test the performance of LSA. The results suggest that, a value $r \approx 400$ provides the best performance, and there is something of an "island of stability" in the r = 300 to 500 range. According to this

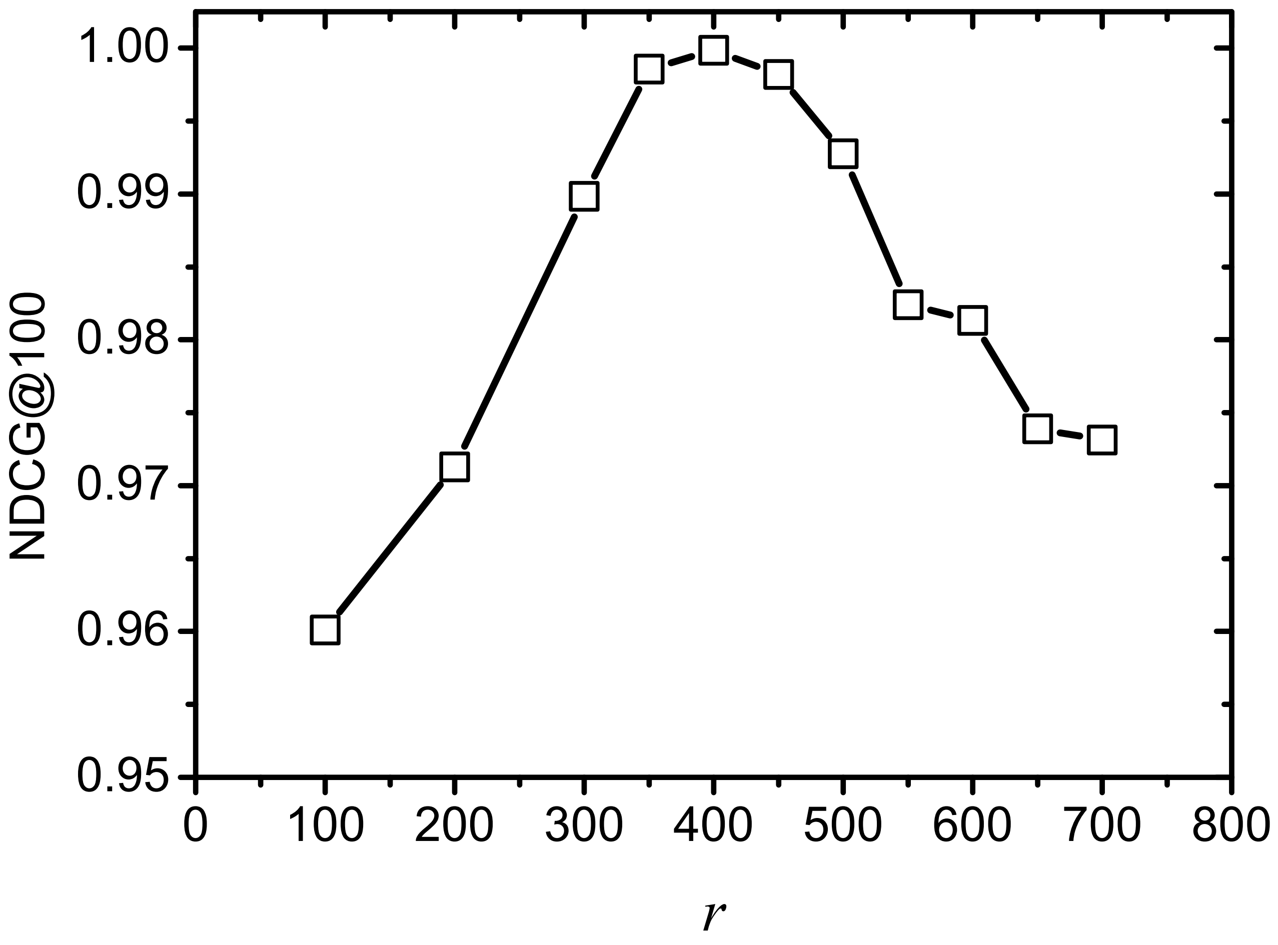

Fig 5. NDCG on varying *r*.

https://doi.org/10.1371/journal.pone.0177523.g005

conclusion, we set parameter $r = 400$ to test both LSA and ILSA in our experiments. Other parameter settings of the comparison method are implemented strictly following the literature. We input 10 queries that consists of two keywords to test the NDCG value and the time cost of on-line query processing. In order to accurately test the execution time of query processing, we process each query with 10 runs, and then average the total time cost.

Effectiveness

In this section, we observe effectiveness of ILSA through testing the NDCG value by setting different threshold θ, and then choose different k and r to observe the NDCG value on a fixed θ.

Fig 3 shows NDCG values on varying threshold θ and the interval is 0.001, where k is set as 100. From $\theta = 0$ to 0.010, we observe that NDCG value decreases with θ increasing, this is because higher θ would lead to more accuracy loss, which is consistent with our previous discussions in Theorem 1. We also observe that the accuracy loss of ILSA before $\theta = 0.01$ is not too much, which suggests a good ranking quality of our approach.

Fig 4 shows the NDCG change on different position k, where $\theta = 0.001, 0.005, 0.010$, and ILSA(0.001), ILSA(0.005), ILSA(0.010) represent the ILSA algorithms at $\theta = 0.001, 0.005, 0.010$

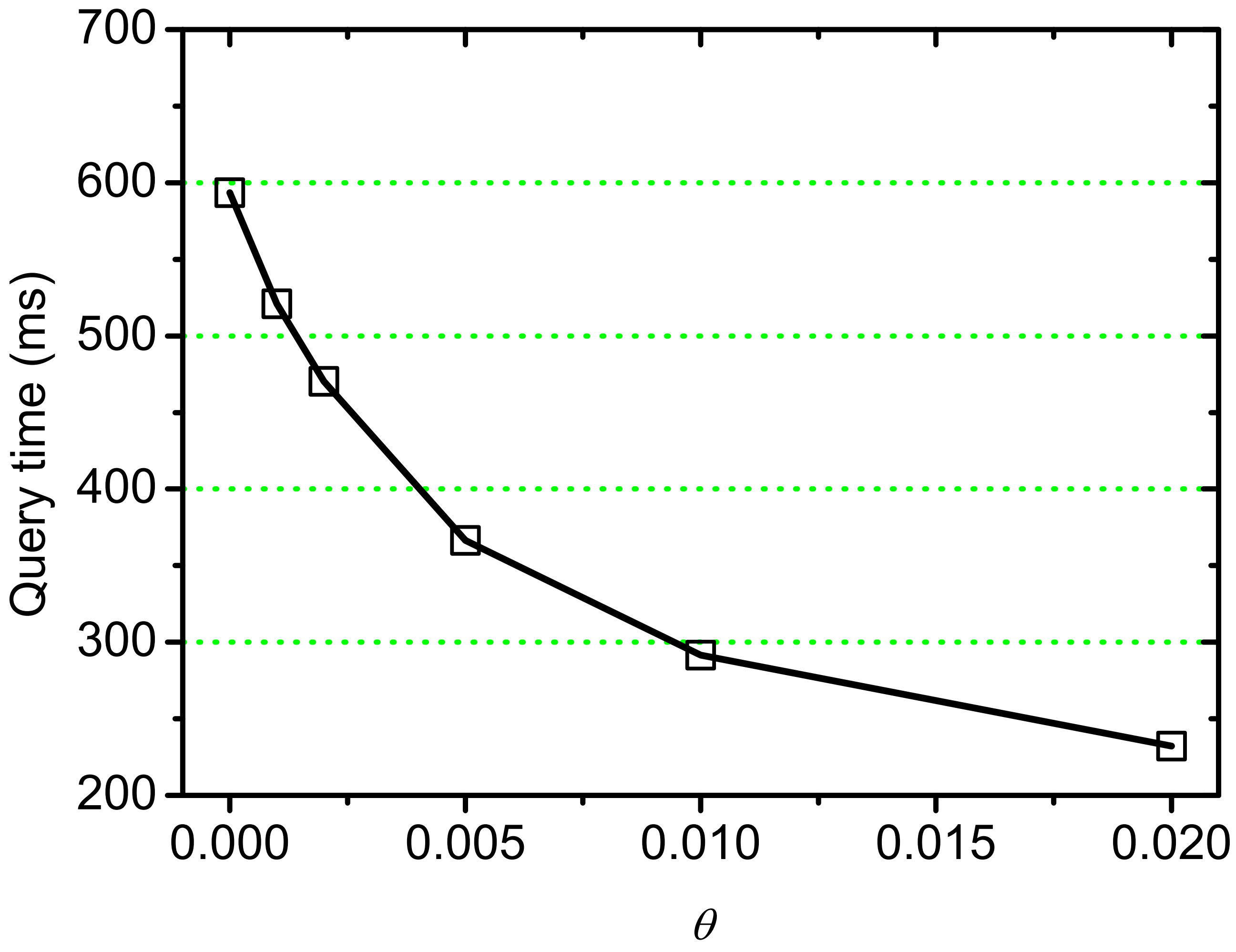

Fig 6. Query processing time on varying *θ*.

https://doi.org/10.1371/journal.pone.0177523.g006

respectively. With k increasing, we find that the curve of ILSA(0.001) is nearly horizontal, since the accuracy loss is very minor; and the NDCG values of ILSA(0.005), ILSA(0.010) shown a upward generally as k increases, this is because some similar documents are lost when setting a higher threshold θ. And these similar documents are obtained again as k increases, which increases the accuracy loss. At each position k, the NDCG value of ILSA(0.001) is always close to 1, and ILSA(0.005) is lower than ILSA(0.001) and higher than ILSA(0.010), since higher θ leads to more accuracy loss, which is consistent with the result in Fig 3. The NDCG of both LSA and ILSA(0) are always 1 at each position k, which are not repeatedly shown in our experiment.

Fig 5 shows the NDCG change of ILSA on varying rank r, where $k = 100$ and $\theta = 0.001$. From this result, we observe that the NDCG increases rapidly from $r = 100$ to 350, this is because more informative terms are contained in similarity computation when increasing r, which consequently increases the effectiveness of the returned rankings. From $r = 350$ to 450, the NDCG scores are relatively higher and stable, since the number of informative terms are suitable and the noisy terms are not too many. After $r = 450$, the NDCG value shows a downward trend, since the number of noisy terms are increased when r is set too big, which also affects the returned rankings. This result demonstrates that the returned rankings of ILSA are

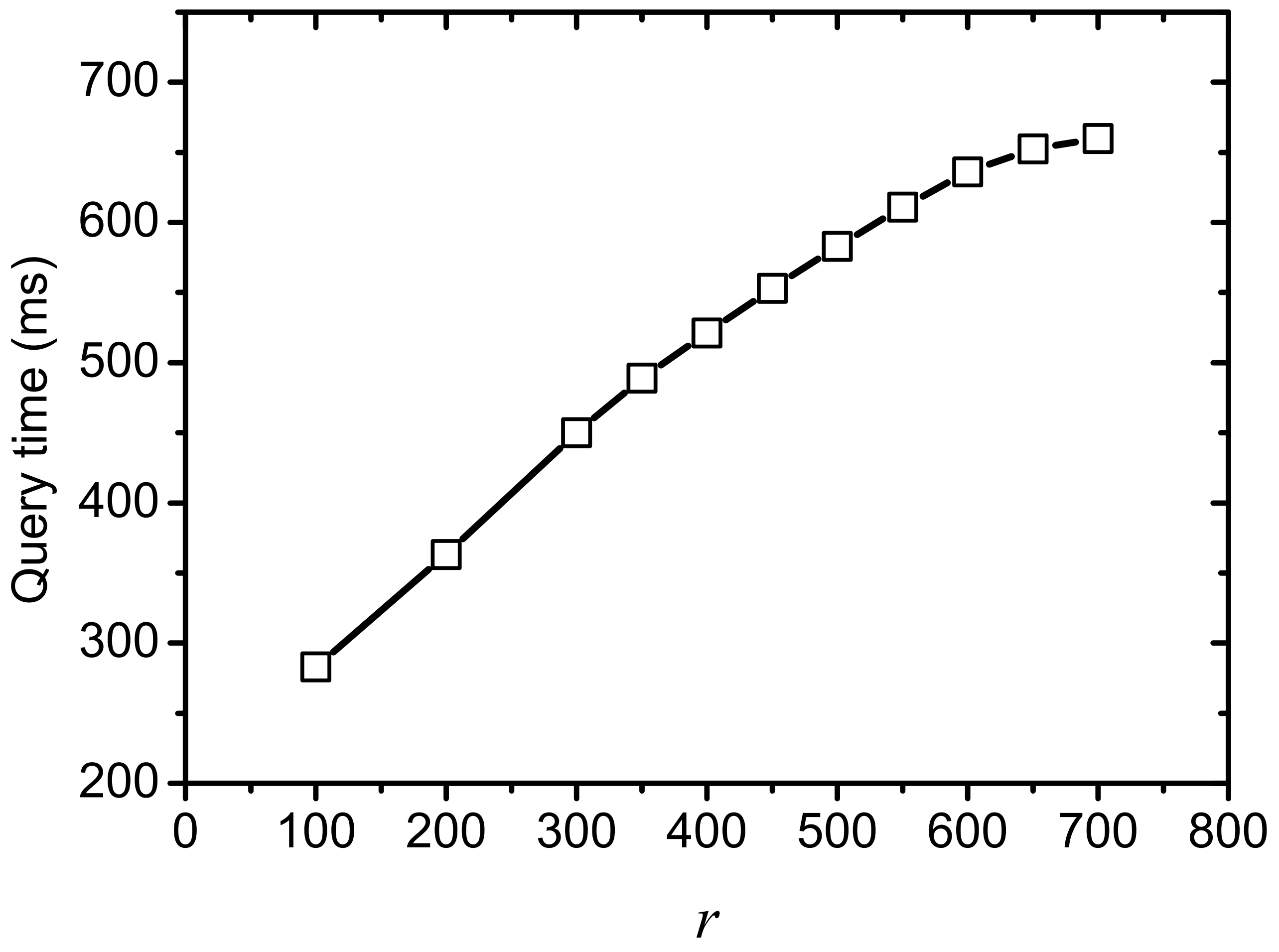

Fig 7. Query processing time on varying *r*.

https://doi.org/10.1371/journal.pone.0177523.g007

affected evidently by r, and the effectiveness would be decreased when r is set too big or too small.

Efficiency

Fig 6 shows the execution time of on-line query processing on varying θ, where $k = 100$. From this result, we observe that the time cost decreases with θ increasing, this is because the index nodes corresponds to the partial similarities lower than threshold θ are skipped when building partial index, and subsequently the searching space of on-line query processing is reduced.

Fig 7 shows the time cost of on-line query processing on varying rank r, where $k = 100$ and $\theta = 0.001$. We observe that the execution time of on-line query processing increases with r increasing, this is because more operations on transformation from the query into pseudo document are involved during on-line query processing. And the incremental time becomes smaller as gradually as r increases, since the size of the document set corresponds to each term in partial index is reduced, which reduces the operations for checking candidates during on-line query processing.

Fig 8 shows the execution time of on-line query processing on varying k, where $\theta = 0, 0.001, 0.005, 0.010$ and $k = 50, 100, 150, \ldots, 450$. From this result, we observe that the incremental

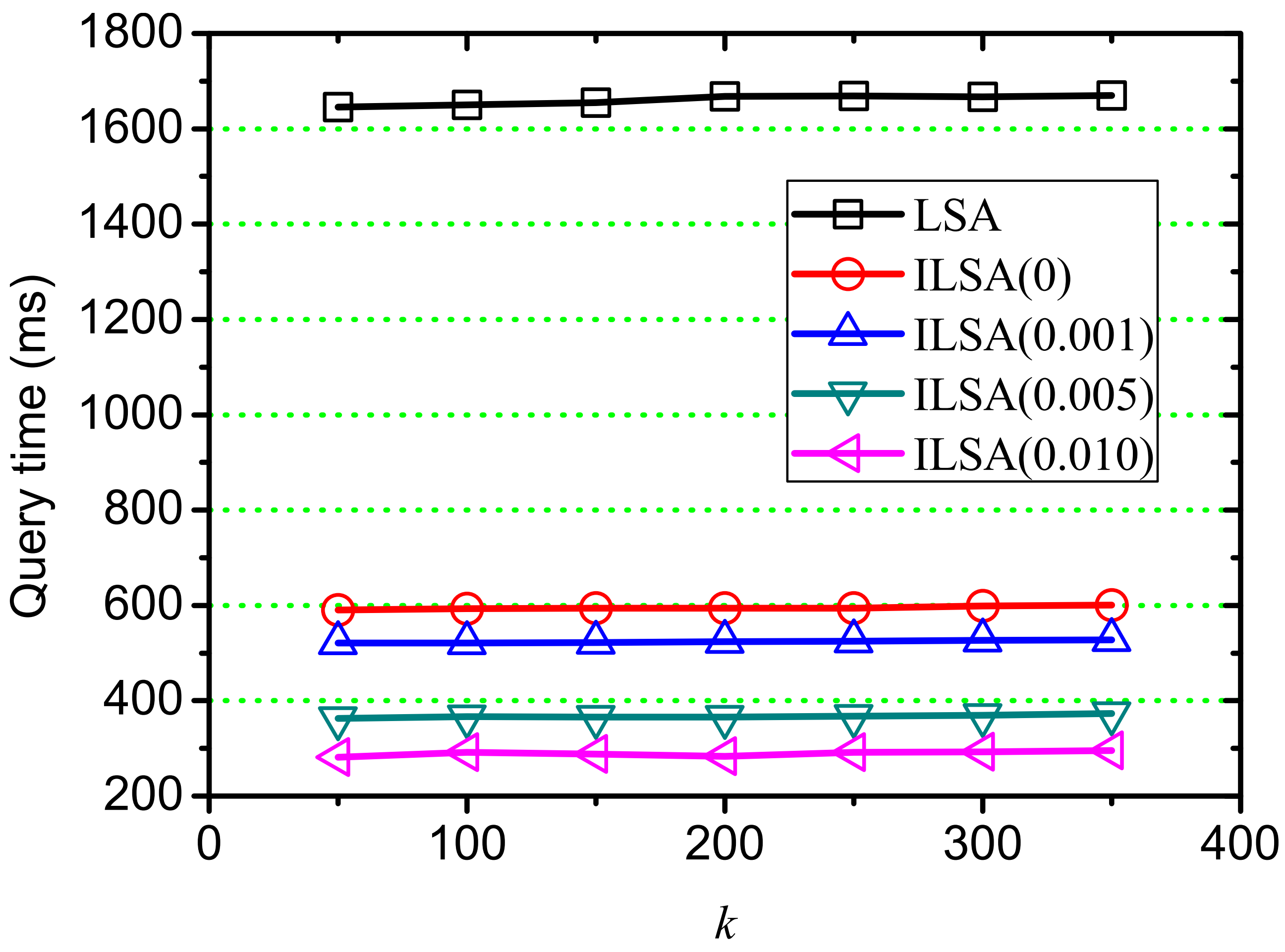

Fig 8. Query processing time on varying *k*.

https://doi.org/10.1371/journal.pone.0177523.g008

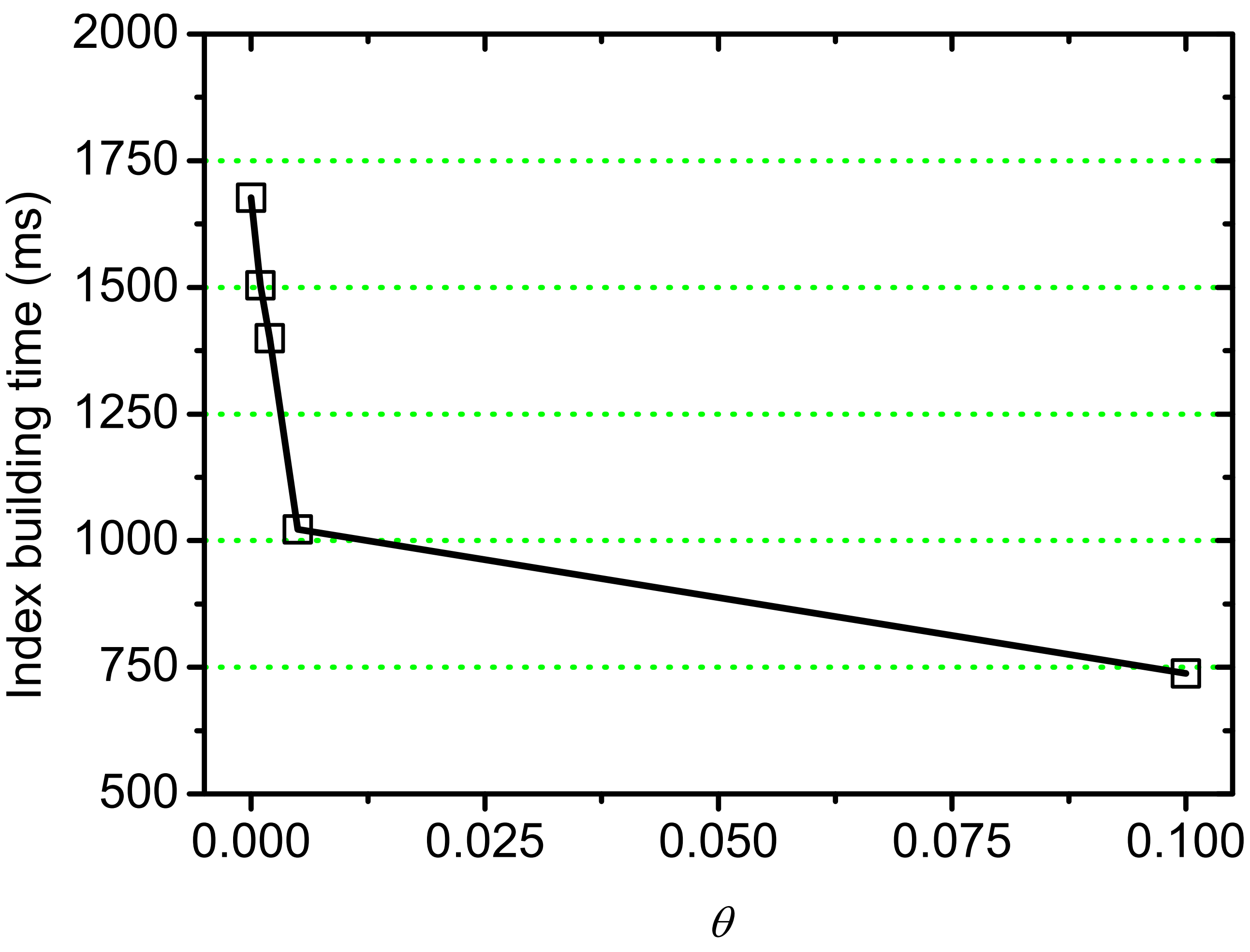

Fig 9. Index building time on varying *θ*.

https://doi.org/10.1371/journal.pone.0177523.g009

time is very minor as *k* increases, this is because the time cost of on-line query processing is mainly affected by the similarity computation between query and candidates and the transformation from the query into the pseudo document vector. Both of these two steps account for a large proportion of the time cost during on-line query processing. ILSA(0.010) is the most efficient method, this is because the searching space is reduced during on-line query processing when setting a higher θ, which is consistent with the result in Fig 6. Generally, our proposed ILSA is more efficient than LSA at different position *k*, which demonstrates the improvement on efficiency of our proposed ILSA.

Fig 9 shows the time cost for building partial index on varying threshold θ. We observe that the time cost for building partial index decreases with threshold θ increasing, this is because the operations for creating index nodes are saved by skipping the partial similarities lower than θ, which is consistent with the previous discussions on complexity analysis. This result demonstrates that the additional time cost in preprocessing stage for building partial index is very low, which would benefit some researches on semantic analysis in real applications.

Fig 10 shows the time cost for building partial index on varying rank *r*, where $k = 100$ and $\theta = 0.001$. From this figure, we find that the time cost of index building increases linearly with *r* increasing, since a bigger *r* can increase the size of SVD matrices and consequently increase

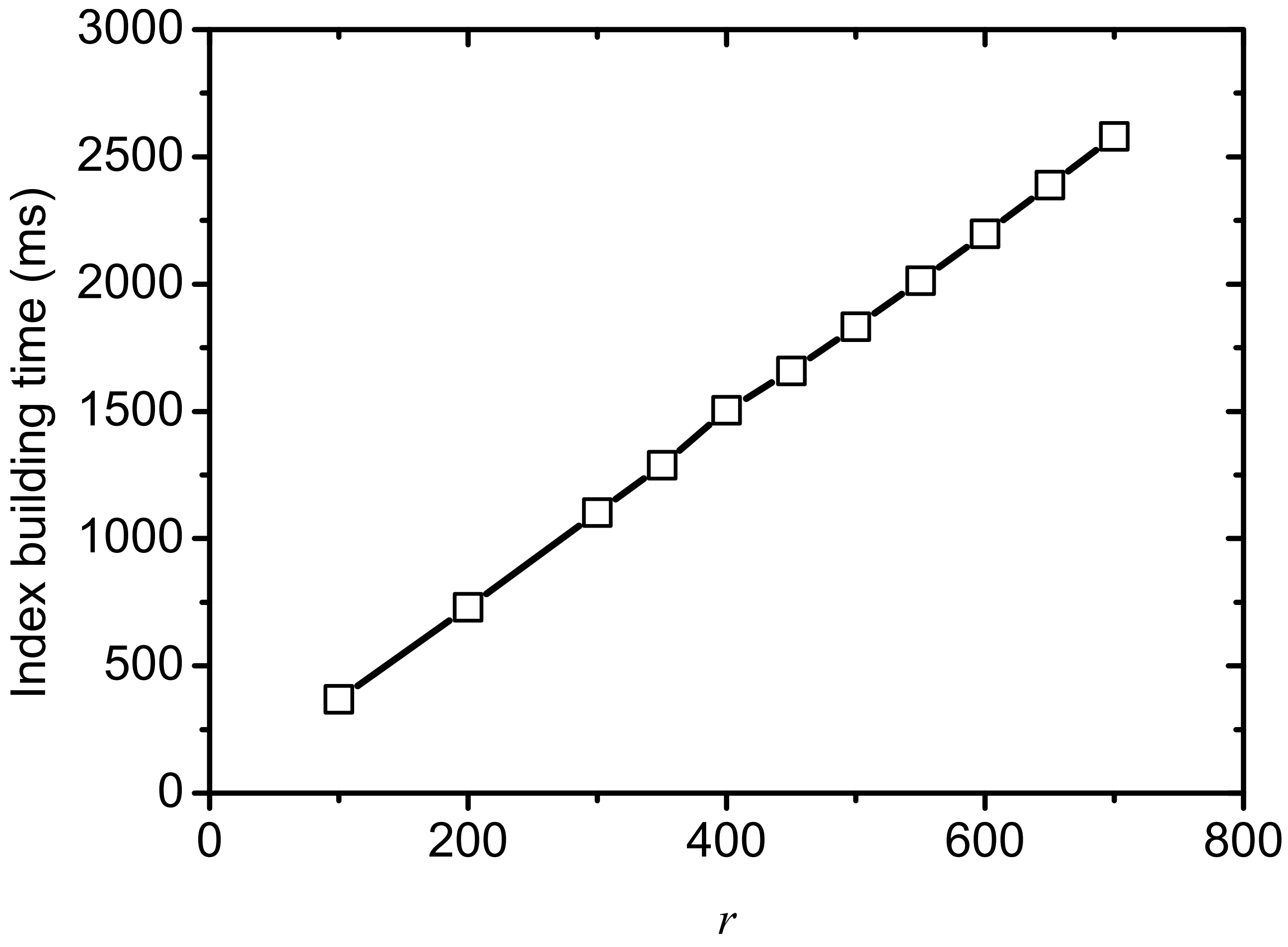

Fig 10. Index building time on varying *r*.

https://doi.org/10.1371/journal.pone.0177523.g010

access operations to these matrices, which is consistent with the analysis on time complexity in Algorithm 1.

Scalability

Fig 11 shows the execution time of on-line query processing on different document scale N, where $\theta = 0, 0.001, 0.005, 0.010$ respectively. We observe that the query processing time increases when document scale grows large, since the incremental documents increase the searching space over partial index during on-line query processing. We also observe that the execution time of ILSA(0) is higher than others, and ILSA(0.010) is the most efficient one, since the operations for computing similarities and checking candidates are reduced by setting a higher threshold θ.

Fig 12 shows the index building time on different document scale N, where $\theta = 0, 0.001, 0.005, 0.010$ respectively. From this figure, we observe that the time cost for building index increases linearly with N increasing, since more access operations on matrices of SVD are involved in the index building process. In practice, although the running time is significantly higher than the query processing time at each document scale, it is acceptable in real applications since the index is built in off-line stage.

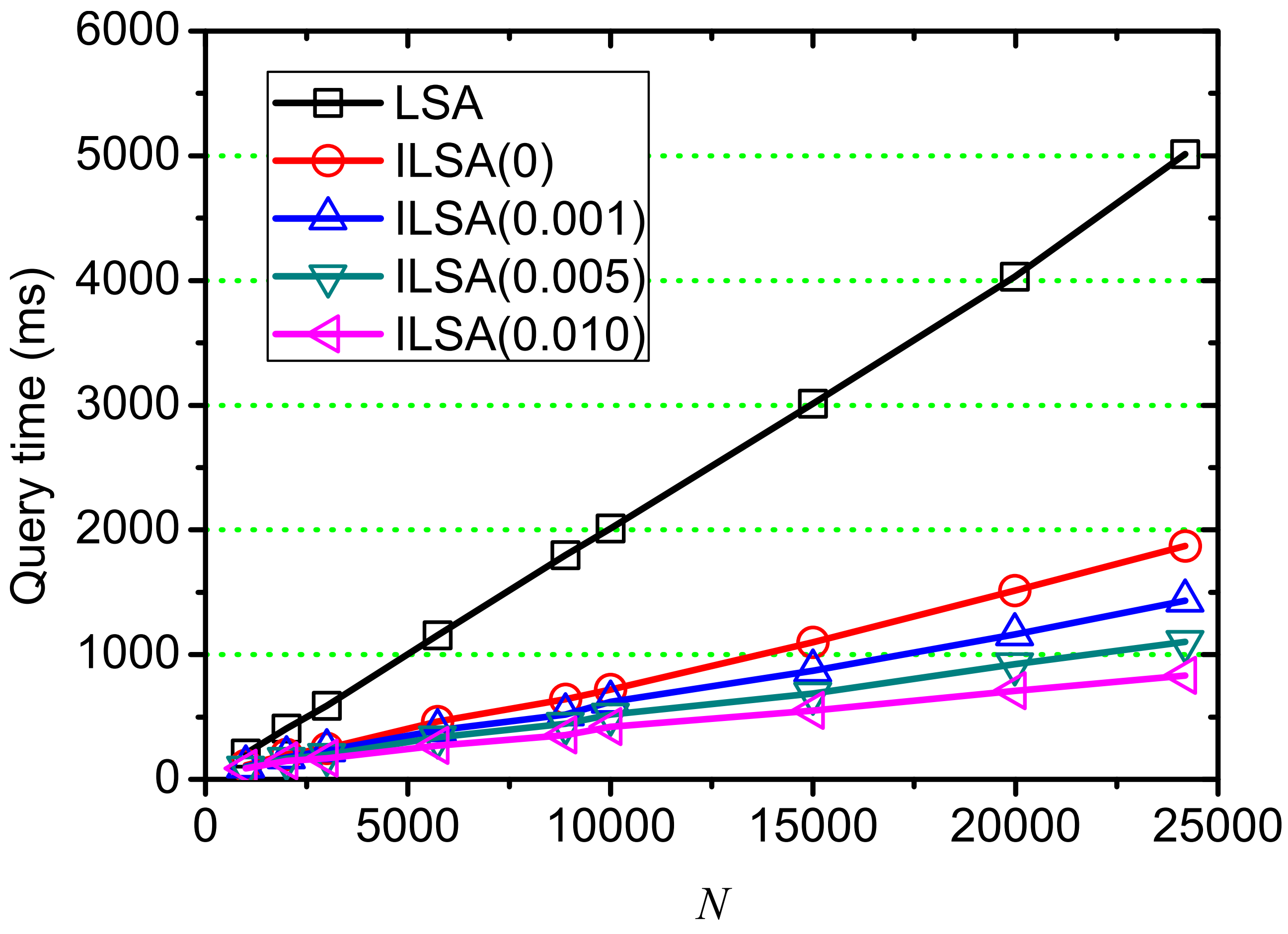

Fig 11. On-line query processing time on varying *N*.

https://doi.org/10.1371/journal.pone.0177523.g011

Fig 13 shows the NDCG change on different document scale N, where $\theta = 0.001, 0.005, 0.010$. We find that the NDCG value of ILSA(0.001) is always close to 1 on varying N and the change is minor, which shows good performance when searching similar documents. The NDCG value of ILSA(0.005) shows a minor downward trend with N increasing, since the candidate set increases when increasing document scale, which subsequently increases the number of similar documents to the given query, but the similar documents should be returned are lost when setting a bigger θ. The downward trend of ILSA(0.010) is more evident than both ILSA(0.001) and ILSA(0.005) as N increases, since $\theta = 0.010$ leads to more accuracy loss when compared to $\theta = 0.001, 0.005$. We also find that the curve of ILSA(0.010) of is evidently lower than both ILSA(0.001) and ILSA(0.005), this is because the effectiveness of the returned rankings would be decreased when setting a higher threshold θ, which is consistent with the result in Fig 3.

Discussion

This paper introduced an index-based query processing algorithm ILSA for efficiently finding similar documents in large document datasets. Compared to the LSA algorithm, ILSA searches the documents over a designed partial index that is derived from the SVD of the

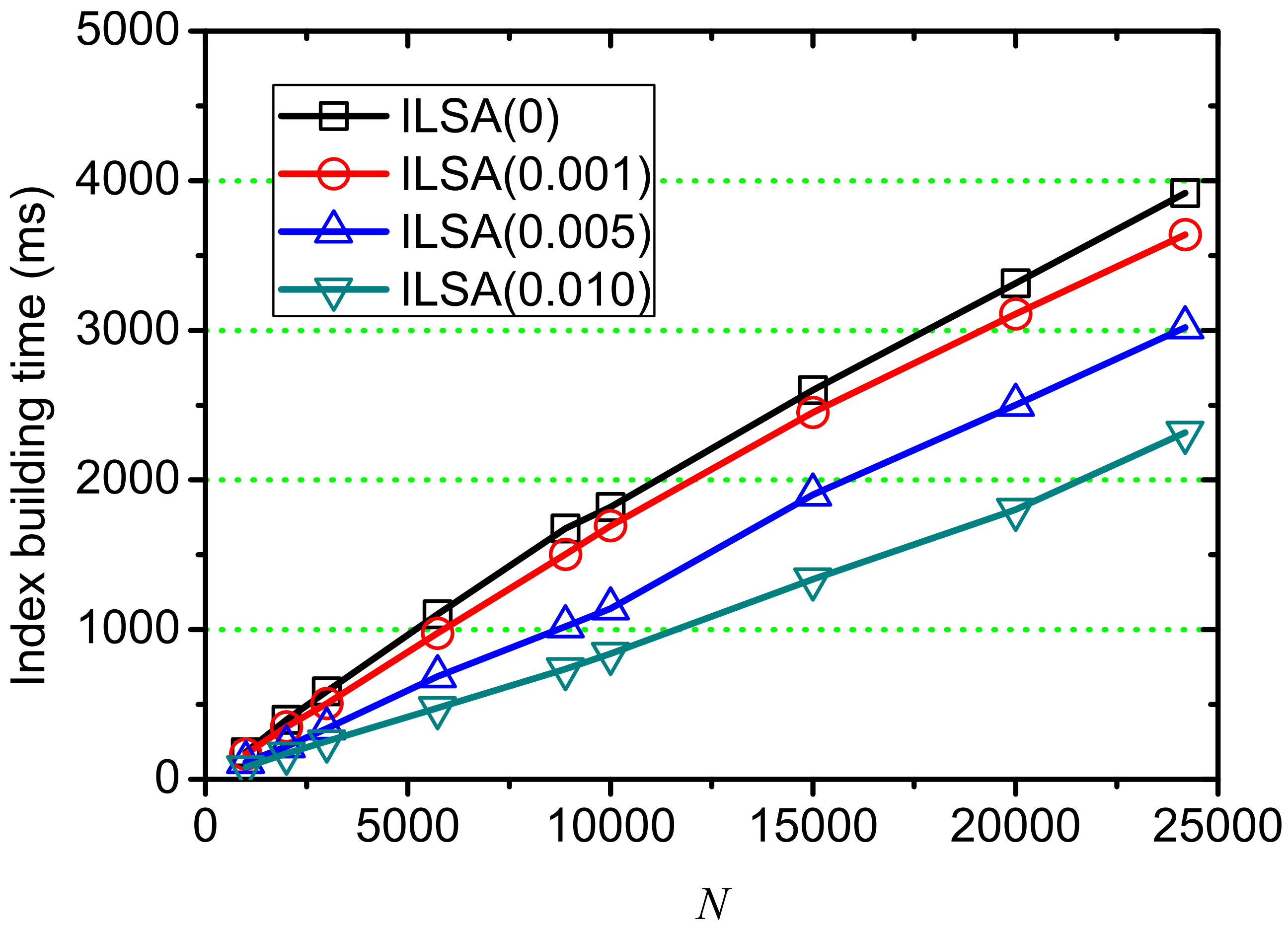

Fig 12. Index building time on varying *N*.

https://doi.org/10.1371/journal.pone.0177523.g012

term-document matrix, and the searching space can be reduced by skipping the partial similarities lower than a given threshold. ILSA reduces the time cost of on-line query processing by pruning the candidate documents that are not promising and skipping the operations that make little contribution to similarity scores, which shows better performance than LSA, and the accuracy loss is under controlled by tuning the threshold. Empirical studies on DBLP through comparison with LSA demonstrate the effectiveness and efficiency of our approach.

There are some directions in our future work. First, ILSA is on the static datasets, and the dynamic datasets are not considered. Accordingly, we will study on how to building a dynamical partial index for the dynamic term set and document set by integrating existing incremental LSA algorithm [61, 62] and incremental SVD algorithm [63–65]. Second, our approach does not pay attention to the transformation process from query into pseudo document which involves lots of unnecessary operations on entries of lower values and increases the execution time of on-line query process. To further reduce the time cost of on-line query process, we plan to optimize the transformation process by skipping the entries of lower values in the SVD matrices, and further optimize the similarity computation between query and candidate documents by skipping the entries of lower values in the vector of pseudo document.

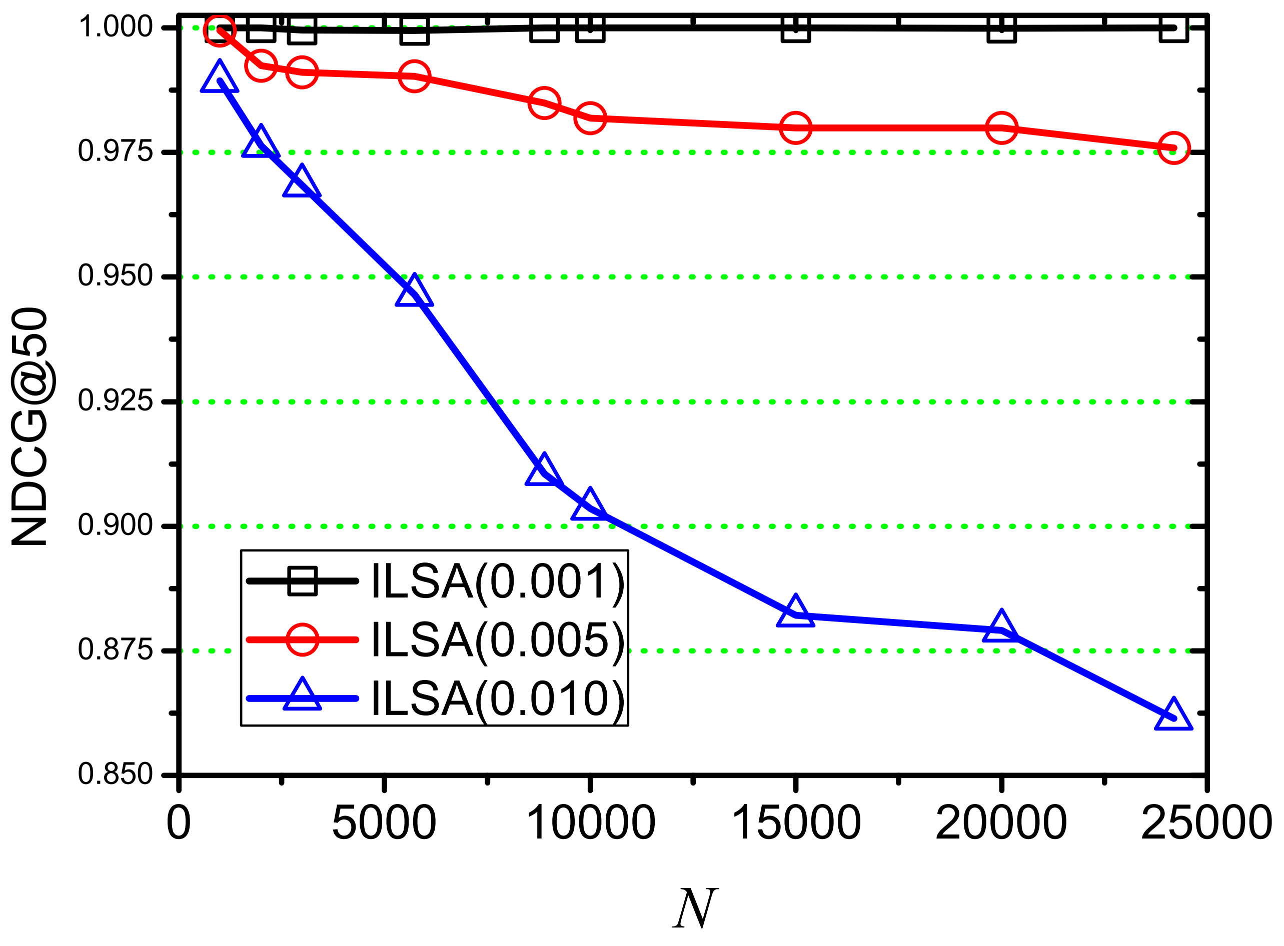

Fig 13. NDCG on varying *N*.

https://doi.org/10.1371/journal.pone.0177523.g013

Acknowledgments

This work was supported by Natural Science Foundation of Shanghai grant 16ZR14228, http://www.stcsm.gov.cn/; Innovation Program of Shanghai Municipal Education Commission grants 15ZZ073 and 15ZZ074, http://www.shmec.gov.cn/; and Training Project of University of Shanghai for Science and Technology grant 16HJPY-QN04, http://www.usst.edu.cn/. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Author Contributions

Conceptualization: MZ PL.

Data curation: MZ PL.

Formal analysis: MZ PL.

Funding acquisition: MZ WW.

Investigation: MZ PL.

Methodology: MZ PL WW.

Project administration: MZ PL WW.

Resources: MZ PL WW.

Software: MZ PL.

Supervision: WW.

Visualization: MZ.

Writing – original draft: MZ PL WW.

References

1. Deerwester SC, Dumais ST, Landauer TK, Furnas GW, Harshman RA. Indexing by Latent Semantic Analysis. JASIS. 1990; 41(6):391–407.
2. Zhang W, Yoshida T, Tang X. TFIDF, LSI and multi-word in information retrieval and text categorization. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Singapore, 12–15 October 2008; 2008. p. 108–113.
3. Landauer TK, Foltz PW, Laham D. An Introduction to Latent Semantic Analysis. Discourse Processes. 1998; 25:259–284. https://doi.org/10.1080/01638539809545028
4. Mirzal A. Clustering and latent semantic indexing aspects of the singular value decomposition. IJIDS. 2016; 8(1):53–72. https://doi.org/10.1504/IJIDS.2016.075790
5. Hofmann T. Probabilistic Latent Semantic Indexing. In: SIGIR; 1999. p. 50–57.
6. Hofmann T. Probabilistic Latent Semantic Analysis. CoRR. 2013;abs/1301.6705.
7. Blei DM, Ng AY, Jordan MI. Latent Dirichlet Allocation. Journal of Machine Learning Research. 2003; 3:993–1022.
8. Anandkumar A, Foster DP, Hsu DJ, Kakade SM, Liu Y. A Spectral Algorithm for Latent Dirichlet Allocation. Algorithmica. 2015; 72(1):193–214. https://doi.org/10.1007/s00453-014-9909-1
9. Chen J, Li K, Zhu J, Chen W. WarpLDA: a Cache Efficient O(1) Algorithm for Latent Dirichlet Allocation. PVLDB. 2016; 9(10):744–755. https://doi.org/10.14778/2977797.2977801
10. Tang L, Peng S, Bi Y, Shan P, Hu X. A New Method Combining LDA and PLS for Dimension Reduction. PLOS ONE. 2014; 9(5):e96944. https://doi.org/10.1371/journal.pone.0096944 PMID: 24820185
11. Latent Factor Models and Matrix Factorizations. In: Encyclopedia of Machine Learning; 2010. p. 571.
12. Luo X, Zhou M, Xia Y, Zhu Q, Ammari AC, Alabdulwahab A. Generating Highly Accurate Predictions for Missing QoS Data via Aggregating Nonnegative Latent Factor Models. IEEE Trans Neural Netw Learning Syst. 2016; 27(3):524–537. https://doi.org/10.1109/TNNLS.2015.2412037 PMID: 25910255
13. Allioua S, Boufaïda Z. Knowledge Representation Using LSA and DRT Rules for Semantic Search of Documents. In: Networked Digital Technologies—4th International Conference, NDT 2012, Dubai, UAE, April 24–26, 2012. Proceedings, Part I; 2012. p. 297–306.
14. Layfield C, Azzopardi J, Staff C. Experiments with Document Retrieval from Small Text Collections Using Latent Semantic Analysis or Term Similarity with Query Coordination and Automatic Relevance Feedback. In: Semantic Keyword-Based Search on Structured Data Sources—COST Action IC1302 Second International KEYSTONE Conference, IKC 2016, Cluj-Napoca, Romania, September 8–9, 2016, Revised Selected Papers; 2016. p. 25–36.
15. An X, Huang JX. Boosting novelty for biomedical information retrieval through probabilistic latent semantic analysis. In: The 36th International ACM SIGIR conference on research and development in Information Retrieval, SIGIR'13, Dublin, Ireland—July 28–August 01, 2013; 2013. p. 829–832.
16. Ghali BE, Qadi AE. Context-aware query expansion method using Language Models and Latent Semantic Analyses. Knowl Inf Syst. 2017; 50(3):751–762.
17. Nugumanova A, Bessmertny I. Applying the Latent Semantic Analysis to the Issue of Automatic Extraction of Collocations from the Domain Texts. In: Knowledge Engineering and the Semantic Web—4th International Conference, KESW 2013, St. Petersburg, Russia, October 7–9, 2013. Proceedings; 2013. p. 92–101.
18. Nath C, Albaghdadi MS, Jonnalagadda SR. A Natural Language Processing Tool for Large-Scale Data Extraction from Echocardiography Reports. PLOS ONE. 2016; 11(4):e0153749. https://doi.org/10.1371/journal.pone.0153749 PMID: 27124000

19. Huang Y. Conceptually categorizing geographic features from text based on latent semantic analysis and ontologies. Annals of GIS. 2016; 22(2):113–127. https://doi.org/10.1080/19475683.2016.1144648

20. Elghazel H, Aussem A, Gharroudi O, Saadaoui W. Ensemble multi-label text categorization based on rotation forest and latent semantic indexing. Expert Syst Appl. 2016; 57:1–11. https://doi.org/10.1016/j.eswa.2016.03.041

21. Franceschini A, Lin J, von Mering C, Jensen LJ. SVD-phy: improved prediction of protein functional associations through singular value decomposition of phylogenetic profiles. Bioinformatics. 2016; 32(7):1085–1087. https://doi.org/10.1093/bioinformatics/btv696 PMID: 26614125

22. Zhang W, Xiao F, Li B, Zhang S. Using SVD on Clusters to Improve Precision of Interdocument Similarity Measure. Comp Int and Neurosc. 2016; 2016:1096271:1–1096271:11. https://doi.org/10.1155/2016/1096271 PMID: 27579031

23. Li C, Han J, He G, Jin X, Sun Y, Yu Y, et al. Fast computation of SimRank for static and dynamic information networks. In: EDBT, Lausanne, Switzerland; 2010. p. 465–476.

24. Ghoshdastidar D, Dukkipati A. Spectral Clustering Using Multilinear SVD: Analysis, Approximations and Applications. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25–30, 2015, Austin, Texas, USA.; 2015. p. 2610–2616.

25. Balouchestani M, Krishnan S. Advanced K-means clustering algorithm for large ECG data sets based on a collaboration of compressed sensing theory and K-SVD approach. Signal, Image and Video Processing. 2016; 10(1):113–120.

26. Wu X, Yang Z, Hu J, Peng J, He P, Zhou J. Diffusion-Weighted Images Superresolution Using High-Order SVD. Comp Math Methods in Medicine. 2016; 2016:3647202:1–3647202:9. https://doi.org/10.1155/2016/3647202 PMID: 27635150

27. Brand M. Fast Online SVD Revisions for Lightweight Recommender Systems. In: Proceedings of the Third SIAM International Conference on Data Mining, San Francisco, CA, USA, May 1–3, 2003; 2003. p. 37–46.

28. Guan X, Li C, Guan Y. Enhanced SVD for Collaborative Filtering. In: Advances in Knowledge Discovery and Data Mining—20th Pacific-Asia Conference, PAKDD 2016, Auckland, New Zealand, April 19–22, 2016, Proceedings, Part II; 2016. p. 503–514.

29. Ntalianis KS, Doulamis AD. Event-complementing Online Human Life Summarization based on Social Latent Semantic Analysis. In: VISAPP 2015—Proceedings of the 10th International Conference on Computer Vision Theory and Applications, Volume 2, Berlin, Germany, 11–14 March, 2015.; 2015. p. 611–622.

30. Dang A, Moh'd A, Islam A, Minghim R, Smit M, Milios EE. Reddit Temporal N-gram Corpus and its Applications on Paraphrase and Semantic Similarity in Social Media using a Topic-based Latent Semantic Analysis. In: COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, December 11–16, 2016, Osaka, Japan; 2016. p. 3553–3564.

31. Kagie M, van der Loos M, van Wezel MC. Including item characteristics in the probabilistic latent semantic analysis model for collaborative filtering. AI Commun. 2009; 22(4):249–265.

32. Yan M, Zhang X, Yang D, Xu L, Kymer JD. A component recommender for bug reports using Discriminative Probability Latent Semantic Analysis. Information & Software Technology. 2016; 73:37–51. https://doi.org/10.1016/j.infsof.2016.01.005

33. Park J, Kang M, Hur J, Kang K. Recommendations for antiarrhythmic drugs based on latent semantic analysis with fc-means clustering. In: 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC 2016, Orlando, FL, USA, August 16–20, 2016; 2016. p. 4423–4426.

34. Boulares M, Jemni M. Learning sign language machine translation based on elastic net regularization and latent semantic analysis. Artif Intell Rev. 2016; 46(2):145–166. https://doi.org/10.1007/s10462-016-9460-3

35. Wang CCN, Lee Y, Sheu PCY, Tsai JJP. Application of Latent Semantic Analysis to Clustering of Cardiovascular Gene Ontology. In: 16th IEEE International Conference on Bioinformatics and Bioengineering, BIBE 2016, Taichung, Taiwan, October 31—November 2, 2016; 2016. p. 363–368.

36. Zeng X, Ding N, Zou Q. Latent factor model with heterogeneous similarity regularization for predicting gene-disease associations. In: IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2016, Shenzhen, China, December 15–18, 2016; 2016. p. 682–687.

37. Zeng X, Zhang X, Zou Q. Integrative approaches for predicting microRNA function and prioritizing disease-related microRNA using biological interaction networks. Briefings in Bioinformatics. 2016; 17(2):193–203. https://doi.org/10.1093/bib/bbv033 PMID: 26059461

38. Zou Q, Li J, Song L, Wang G. Similarity computation strategies in the microRNA-disease network: a survey. Brief Funct Genomics. 2016; 15(1):55–64. https://doi.org/10.1093/bfgp/elv024 PMID: 26134276

39. González J, Muñoz A, Martos G. Asymmetric latent semantic indexing for gene expression experiments visualization. J Bioinformatics and Computational Biology. 2016; 14(4):1–18. https://doi.org/10.1142/S0219720016500232 PMID: 27427382

40. Roy S, Curry BC, Madahian B, Homayouni R. Prioritization, clustering and functional annotation of MicroRNAs using latent semantic indexing of MEDLINE abstracts. BMC Bioinformatics. 2016; 17(S-13):350. https://doi.org/10.1186/s12859-016-1223-2 PMID: 27766940

41. Zhang Y, Yi D, Wei B, Zhuang Y. A GPU-accelerated non-negative sparse latent semantic analysis algorithm for social tagging data. Inf Sci. 2014; 281:687–702. https://doi.org/10.1016/j.ins.2014.04.047

42. Ye Y, Gong S, Liu C, Zeng J, Jia N, Zhang Y. Online belief propagation algorithm for probabilistic latent semantic analysis. Frontiers of Computer Science. 2013; 7(4):526–535. https://doi.org/10.1007/s11704-013-2360-7

43. Drinea E, Drineas P, Huggins P. A randomized singular value decomposition algorithm for image processing. In: Panhellenic Conference on Informatics (PCI); 2001.

44. Brand M. Incremental Singular Value Decomposition of Uncertain Data with Missing Values. In: ECCV (1); 2002. p. 707–720.

45. Levy A, Lindenbaum M. Sequential Karhunen-Loeve basis extraction and its application to images. IEEE Transactions on Image Processing. 2000; 9(8):1371–1374. https://doi.org/10.1109/83.855432 PMID: 18262974

46. Holmes MP, Gray AG, Jr CLI. QUIC-SVD: Fast SVD Using Cosine Trees. In: Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8–11, 2008; 2008. p. 673–680.

47. Drmac Z, Veselic K. New Fast and Accurate Jacobi SVD Algorithm. I. SIAM J Matrix Analysis Applications. 2007; 29(4):1322–1342. https://doi.org/10.1137/050639193

48. Drmac Z, Veselic K. New Fast and Accurate Jacobi SVD Algorithm. II. SIAM J Matrix Analysis Applications. 2007; 29(4):1343–1362. https://doi.org/10.1137/05063920X

49. Ait-Haddou R, Barton M. Constrained multi-degree reduction with respect to Jacobi norms. Computer Aided Geometric Design. 2016; 42:23–30. https://doi.org/10.1016/j.cagd.2015.12.003

50. Strobach P. The fast householder Bi-SVD subspace tracking algorithm. Signal Processing. 2008; 88(11):2651–2661. https://doi.org/10.1016/j.sigpro.2008.05.004

51. Zhou L, Li C. Outsourcing Eigen-Decomposition and Singular Value Decomposition of Large Matrix to a Public Cloud. IEEE Access. 2016; 4:869–879. https://doi.org/10.1109/ACCESS.2016.2535103

52. Menon V, Du Q, Fowler JE. Fast SVD With Random Hadamard Projection for Hyperspectral Dimensionality Reduction. IEEE Geosci Remote Sensing Lett. 2016; 13(9):1275–1279. https://doi.org/10.1109/LGRS.2016.2581172

53. Joachims T. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. In: Proceedings of the Fourteenth International Conference on Machine Learning (ICML 1997), Nashville, Tennessee, USA, July 8–12, 1997; 1997. p. 143–151.

54. Qin P, Xu W, Guo J. A novel negative sampling based on TFIDF for learning word representation. Neurocomputing. 2016; 177:257–265. https://doi.org/10.1016/j.neucom.2015.11.028

55. Domeniconi G, Moro G, Pasolini R, Sartori C. A Study on Term Weighting for Text Categorization: A Novel Supervised Variant of tf.idf. In: DATA 2015—Proceedings of 4th International Conference on Data Management Technologies and Applications, Colmar, Alsace, France, 20–22 July, 2015.; 2015. p. 26–37.

56. Chen K, Zhang Z, Long J, Zhang H. Turning from TF-IDF to TF-IGM for term weighting in text classification. Expert Syst Appl. 2016; 66:245–260. https://doi.org/10.1016/j.eswa.2016.09.009

57. Zhang M, Hu H, He Z, Gao L, Sun L. Efficient link-based similarity search in web networks. Expert Syst Appl. 2015; 42(22):8868–8880. https://doi.org/10.1016/j.eswa.2015.07.042

58. Zhang M, Hu H, He Z, Wang W. Top-k similarity search in heterogeneous information networks with x-star network schema. Expert Syst Appl. 2015; 42(2):699–712. https://doi.org/10.1016/j.eswa.2014.08.039

59. Järvelin K, Kekäläinen J. Cumulated gain-based evaluation of IR techniques. ACM Trans Inf Syst. 2002; 20(4):422–446.

60. Bradford RB. An empirical study of required dimensionality for large-scale latent semantic indexing applications. In: Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008, Napa Valley, California, USA, October 26–30, 2008; 2008. p. 153–162.

61. Zhang M, Hao S, Xu Y, Ke D, Peng H. Automated Essay Scoring Using Incremental Latent Semantic Analysis. JSW. 2014; 9(2):429–436. https://doi.org/10.4304/jsw.9.2.429-436

62. Çelikkanat H, Orhan G, Pugeault N, Guerin F, Sahin E, Kalkan S. Learning Context on a Humanoid Robot using Incremental Latent Dirichlet Allocation. IEEE Trans Cognitive and Developmental Systems. 2016; 8(1):42–59.

63. Lee M, Choi C. Incremental (N) -Mode SVD for Large-Scale Multilinear Generative Models. IEEE Trans Image Processing. 2014; 23(10):4255–4269. https://doi.org/10.1109/TIP.2014.2346012 PMID: 25122567

64. Iwen MA, Ong BW. A Distributed and Incremental SVD Algorithm for Agglomerative Data Analysis on Large Networks. CoRR. 2016;abs/1601.07010. https://doi.org/10.1137/16M1058467

65. Balzano L, Wright SJ. On GROUSE and Incremental SVD. CoRR. 2013;abs/1307.5494. https://doi.org/10.1109/CAMSAP.2013.6713992