Efficient Set-Correlation Operator Inside Databases

Shaoxu Song
The Hong Kong University of Science and Technology, Hong Kong
sshaoxu@cse.ust.hk

Lei Chen
The Hong Kong University of Science and Technology, Hong Kong
leichen@cse.ust.hk

ABSTRACT

Large scale of short text records are now prevalent, such as news highlights, scientific paper citations, and posted messages in a discussion forum, which are often stored as set records in (hidden) databases. Many interesting information retrieval tasks are correspondingly raised on the correlation query over these short text records, such as finding hot topics over news highlights and searching related scientific papers on a certain topic. However, current *relational database management systems* (RDBMS) do not directly provide support on set correlation query. Thus, in this paper, we address both the *effectiveness* and *efficiency* issues of set correlation query over set records in databases. First, we present a framework of set correlation query inside databases. To our best knowledge, only the Pearson's correlation can be implemented to construct token correlations by using RDBMS facilities. Thereby, we propose a novel correlation coefficient to extend Pearson's correlation, and provide a pure-SQL implementation inside databases. We further propose optimal strategies to set up correlation filtering threshold, which can greatly reduce the query time. Our theoretical analysis proves that, with a proper setting of filtering threshold, we can improve the query efficiency with a little effectiveness loss. Finally, we conduct extensive experiments to show the effectiveness and efficiency of proposed correlation query and optimization strategies.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications

General Terms

Measurement, Performance

1. INTRODUCTION

Short text records become more and more popular, which are often stored as set records in (hidden) databases. For example, in many news agencies web sites, instead of offering full text documents, often, news highlights are shown on the front page to attract users' interests. These news short highlights are examples of set records. Given another example, in many scientific paper citation web sites, such as Citeseer and ACM Portal, paper citations are given in the form of short texts. Other examples of short texts as set records include users' text inputs in the logs of a search engine and reply messages in a discussion forum. Efficient set similarity operators over such set records have been well studied [2, 15, 22].

With the emerging of these short text records, a very useful yet interesting query, *correlation query*, has been introduced in many applications to find the *correlated* records (e.g., with similar topics [36]). For example, in order to detect hot topics among the news highlights from different news agencies (e.g. BBC, CNN, and AOL), we can conduct a correlation query for each highlight (a short set record) to find a number of other highlights in the database that are correlated to the query on similar topics. The set record that has the most number of correlated records can be treated as the hot topic in the current database. Given another example application on managing scientific literature data, literature records can be collected from different personal web pages, official publication web sites (IEEE or ACM), and conference proceedings. As shown in Table 1, we usually store paper citations as set records in databases. Again, it is promising to conduct a query of correlated records over set records in this database. For example, suppose that a fresh postgraduate student want to search papers related to "record linkage". A comprehensive *correlation query* result is expected to return the literatures in "duplicate record detection", "reference reconciliation" and "entity resolution" as well. Motivated by these interesting applications, we study the correlation query over large scale set records in databases.

Unfortunately, current *relational database management systems* (RDBMS) do not directly provide support on correlation query. A novel framework should be studied by using current RDBMS facilities to support the correlation query on set records effectively and efficiently inside databases. We emphasize the importance of implementing correlation query inside RDBMS, since it can be naturally extended to *correlation join* over set records as well. With the implementation in RDBMS, correlation query can benefit in the following aspects. First, we can utilize the optimization facilities provided by RDBMS engine, such as join order selection, to automatically achieve the optimal performance [9, 5, 18]. Second, with the help of RDBMS, we can handle very large

CIKM'10, October 26–30, 2010, Toronto, Ontario, Canada.

Table 1: Set records in a scientific literature database

ID	**Paper** (Set records)
1	Duplicate Record Detection: A Survey; Ahmed K. Elmagarmid et al.; record linkage, data deduplication, entity resolution
2	Record Linkage: Similarity Measures and Algorithms; SIGMOD Tutorial 2006.
3	A latent dirichlet model for unsupervised entity resolution; Lise Getoor; www.cs.umd.edu/~getoor
4	Reference reconciliation in complex information spaces; Xin Dong; record linkage, merge/purge, deduplication, reference matching, object identification, identity uncertainty

size of correlations for real-world vocabularies, for example, consider the vocabulary from the Web. Therefore, in this study, we mainly focus on seeking and querying correlations inside databases.

Contributions. In this paper, motivated by the huge amount of set records stored in databases, we address both the *effectiveness* and *efficiency* issues in the correlation query over set records in databases. To our best knowledge, this is a first attempt to support correlation query over set records inside databases. Our contributions in this paper are summarized as follows:

- We introduce a framework for representing and querying correlations in databases (in Section 3), and present a pure-SQL implementation of correlation query by utilizing the optimization facilities in current RDBMS. By utilizing the facility provided by RDBMS, we can process the correlation query efficiently and address the issue of scalability. Furthermore, the SQL-based implementation of correlation query can be extended easily to support other applications on set records.
- We develop a novel *token inverted correlation* function based on the co-occurrence statistics of tokens in records (in Section 4), which extends and outperforms the Pearson's correlation coefficient in terms of effectiveness.
- Our theoretical analysis proves that with a certain filtering threshold, we can improve query efficiency with a little effectiveness loss (in Section 5). Our extensive experiments also confirm the optimization analysis.
- Our proposed method achieves about 20% higher accuracy of query results and costs little extra runtime compared to the similarity matching query. The proposed approach scales well under large data sizes in the experiments (in Section 6).

The rest of this paper is organized as follows: In Section 3, we introduce a query framework with respect to correlations on the set record, and the corresponding implementation in RDBMS. Section 4 presents the definition of token correlations, as well as the construction of correlations inside databases. Section 5 addresses how to answer correlation queries efficiently, and provides theoretical analysis for optimization issues. Section 6 demonstrates the performance of our approach in the experiments. Finally, we conclude this paper in Section 7.

2. RELATED WORK

Techniques on set similarity operators over set records have been well studied [2, 15, 22], where the matching similarity is mainly considered between set records. Instead of set similarity, in this paper, we mainly focus on operators of set correlation. In order to evaluate the correlated set records, most of the correlation query approaches [7, 19, 38] use the Pearson's correlation coefficient [37] to measure correlations, which can also be implemented in databases.

Recently, Sahami and Heilman [30] measure the correlation between short set records by leveraging web search results to provide greater context for short texts. Thus, the correlation relies heavily on the quantity of search results. Chirita et al. [10] propose the query expansion by using users' personalized information. Several techniques for determining expansion terms from personal documents are introduced, including co-occurrence statistics and external thesauri. Theobald et al. [34] also study the query expansion in a dynamical and incremental way. A priority queue is used for maintaining result candidates and pruning of candidates together with probabilistic estimators of candidate scores. Liu et al. [26] utilized the WordNet to formulate the correlation between word tokens in documents. However, these techniques require knowledge outside databases and can hardly be directly applicable by using current RDBMS facilities. In this paper, instead of using the general domain knowledge outside databases, we are interested in conducting correlation quires inside databases.

The retrieval of small text records as sentence level is also studied in [27, 1, 4, 25, 24]. Li and Croft [25, 24] learn sentence level information patterns from the training data to identify potential answers. Murdock and Croft [28] conduct the sentence retrieval as translations from the query to the results. A parallel corpus has to be exploited for training the translation model. Fung and Lee [13] also develop the correlations of new words for translation, by considering the co-occurrence of other (seed) words in the sentences. Cao et al. [8] develop a dependency model to incorporate the co-occurrence information in language modeling. Unfortunately, the proposed techniques with language models are too complicated to be supported by using the current RDBMS facilities, and again, not directly applicable in our correlation query (as well as join) problem inside databases.

Furthermore, learnable similarity metrics on set records have been investigated in recent studies. For example, Jin et al. [20] propose a supervised term weighting scheme by considering the correlation between word frequency and category information of documents. Bilenko and Mooney [6] compute the comparison similarity vector of two records and classify the vector as similar or not with a similarity value output. Sarawagi et.al. [32] propose an active leaning approach by picking up the most uncertain data which will be labeled manually. These approaches are learning techniques, which need training data sets. Different from the learning approach, in this paper, we mainly focus on the approaches that explore correlations inside the dataset and do not require pre-labeled training data.

Hofmann [16, 17] proposes the *probabilistic latent semantic analysis* (PLSA), which addresses the problem of different words with a similar meaning or the same concept. However, the *probabilistic latent semantic analysis* as an extension of *latent semantic analysis* LSA [12] is a type of dimension reduction techniques. Rather than removing the

query	
qid	tid
q_1	t_1
q_1	t_2
q_1	t_3
...	...
q_1	t_m

correlation		
tid_i	tid_j	w_{ij}
t_1	t_3	w_{13}
t_2	t_3	w_{23}
t_2	t_2	w_{22}
...	...	...
t_i	t_j	w_{ij}

records	
tid	rid
t_2	r_1
t_3	r_1
t_2	r_2
...	...
t_i	r_n

Figure 1: Storage scenario of correlation query

tokens, our correlation-based query enriches the correlations between tokens and finds the correlations between tokens without any class knowledge.

3. SET CORRELATION QUERY

In this work, we consider set records stored in relational databases. This section provides a framework to support correlation query inside databases. Formally, we define the correlation query problem over set records as follows:

PROBLEM 3.1 (CORRELATION QUERY). *Given a query q (a set record with several tokens as well), the* correlation query *returns the records r ranked by $cor(q, r)$, where cor is the* correlation *between q and r.*

3.1 Query Framework

Due to the sparse features of text data, we extend a vertical schema to store set records [5]. The data storage scenario in query framework is illustrated in Figure 1. Specifically, we segment each record into a set of tokens (e.g. words or *q-grams*) and store all the pairs of token *tid* and record *rid* in the table $records(tid, rid)$. As presented in Section 4, we can compute a correlation weight of each token pair in the database. These token correlations corresponding with their weights are represented in the view, $correlation(t_i, t_j, w_{ij})$, where w_{ij} denotes the correlation weight $cor(t_i, t_j)$ between tokens t_i and t_j.

Similarity matching-based query of short text strings in databases has been studied by [11, 14, 9, 3]. The query is directly performed on *records* based on matching tokens between the query and records. As shown in Figure 2(a), we can evaluate the similarity matching query in our framework by conducting a Join operation on the *query* and *records* tables with $query.tid = records.tid$, i.e., the similarity matching query:

$$T_s \leftarrow query \bowtie_{tid} records$$

The similarity matching query generates all candidate records which have common matching tokens with the query, when the results of T_s are grouped by the record *rid*. For each group of a record, we can compute an aggregation score as the matching similarity, e.g., $sim(q, r) = |q \cap r|$.

To answer correlation queries, we have to consider all the pairs of tokens (t_i, t_j) between the query q and record r. Based on the pairs of tokens with correlations, we can determine the correlation of the query q and r. Therefore, as shown in Figure 2(b), we consider the correlations between set records in the *query* and *records* directly one by one. Specifically, for each pair of query and record (q, r), we have to search the *correlation* data frequently, since the

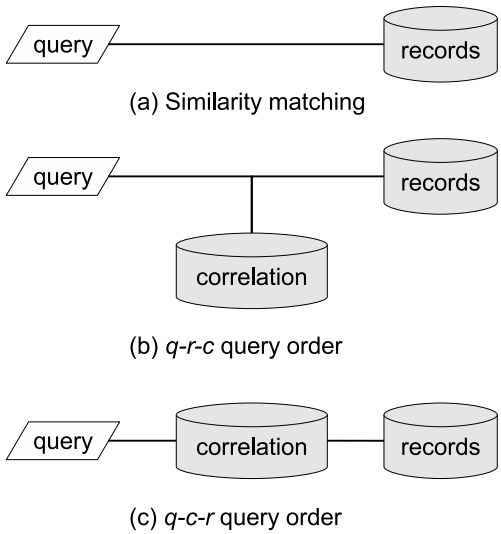

Figure 2: Query framework

correlations between tokens in the query and records are represented in the *correlation* view. In terms of storage in Figure 1, we conduct a Cartesian product on the tables of *query* and *records* to generate a large number of results as candidates. Then, token correlations are performed on these pairs of candidates to select the records with highest correlations, i.e., the $q - r - c$ correlation query order:

$$T_c \leftarrow (query \times records) \bowtie_{tid_i, tid_j} correlation$$

The correlation query with $q - r - c$ order considers all the possible combination of token pairs across the query and each record, and then checks whether the correlations between those token pairs exist. Again, after finding all the candidates of query results in T_c, the efficient facilities in RDBMS, such as `GROUP BY` and `ORDER BY`, can be utilized directly for aggregating and ranking the results.

Moreover, since the data are stored in a relational database, we can automatically optimize the correlation query by utilizing the efficient facilities provided by RDBMS, such as join order selection. Note that the results of Cartesian product are large in size according to the $q - r - c$ correlation query order, where most pairs of tokens might not have correlations. Thus, rather than enumerating all the pairs of tokens between the query and records, we select the tokens with correlations first. In terms of storage in Figure 1, we first perform the Join operation on the table of *query* and *correlation*, rather than the connecting on *query* and *records*. Since the query size is small, the result of the first join step will be small as well. Then, we query the small size output in the *records* like the similarity matching approaches. Consequently, the correlation query is rewritten in the $q - c - r$ order,

$$T'_c \leftarrow (query \bowtie_{tid_i} correlation) \bowtie_{tid_j} records$$

i.e., first generating all the tokens correlated with the tokens in the query, and then performing the efficient matching with *records* based on common (matched) tokens. We use SQL to implement correlation queries, thus, the query optimizer of RDBMS can automatically be used to select the above correct join order based on the join sizes and available indexes.

Note that this framework can be naturally extended to *correlation join* over set records as well. That is, we replace *query* with set records on attribute A_1 in Figure 1, and *records* denote the set records on attribute A_2. Then, the correlation join over A_1 and A_2 can be evaluated, which is out of the scope of this paper.

3.2 Record Correlation

We now define the correlation between two set records. There are many distance functions designed for measuring string similarity, including *matching coefficient*, *Jaccard coefficient*, *Euclidean distance*, *Cosine similarity* and so on [31]. These distance functions consider pairs of matched tokens between two strings, where the relationship between matched tokens are one-to-one. In our correlation query case, instead of matched token pairs, we study correlated token pairs between query q and record r. Note that the correlations between tokens are many-to-many relationship, that is, one token may be correlated with several tokens in the other set record.

Given a query q and a record r, $\mathcal{C}(q,r)$ is used to represent the set of all the pairs of tokens between q and r whose token correlations are greater than 0.

$$\mathcal{C}(q,r) = \{cor(t_i,t_j) \mid t_i \in q, t_j \in r, cor(t_i,t_j) > 0\} \quad (1)$$

where $cor(t_i,t_j)$ denotes the token correlation between t_i and t_j, i.e., w_{ij} in Figure 1. In the following Section 4, we introduce the construction of such token correlations, i.e., either the Pearson's correlation in formula 3 or our proposed inverted correlation in formula 9. We evaluate both token correlation measures through extensive experiments and report the results Section 6.

Consequently, we aggregate all the token correlations $cor(t_i,t_j)$ in the set of $\mathcal{C}(q,r)$ as the record correlation

$$cor(q,r) = \sum_{t_i \in q} \sum_{t_j \in r} cor(t_i,t_j) \quad (2)$$

As mentioned, such record correlation score can be naturally computed by an aggregation query provided by RDBMS, once token correlations $cor(t_i,t_j)$ are obtained.

3.3 Correlation Query Implementation

As presented in the query framework in Section 3.1, we store the query in the table $query(qid)$. The following algorithm aggregates all the correlation weights of each record. The returned results of rid are ordered in descending order according to the scores. Though the algorithm is implemented with $q-c-r$ order, we can rely on the query optimizer of RDBMS to select the optimal join order, most probably the $q-c-r$ order.

```
SELECT SUM(T.Cweight) AS score, D.rid
FROM records AS D,
 (SELECT C.tid2 AS Ctid, C.weight AS Cweight
  FROM correlation AS C, query AS Q
  WHERE C.tid1 = Q.tid) AS T
WHERE D.tid = T.Ctid
GROUP BY D.rid
ORDER BY score DESC
```

In order to evaluate and compare the correlation query and similarity matching query, we also present the implementation of the inverted index-based similarity matching approach. The following algorithm counts all the matching tokens between the query `q` and the record `r` as the similarity score and rank the records in the descending order.

```
SELECT COUNT(*) AS score, r.rid
FROM records AS r, query AS q
WHERE r.tid = q.tid
GROUP BY r.rid
ORDER BY score DESC
```

4. CORRELATION CONSTRUCTION

As the framework illustrated in Figure 1 and 2, the key issue of supporting correlation query is to construct token correlations inside databases. Many sophisticated work on measuring token correlations have been proposed in information retrieval literature. Unfortunately, due to the limitation of facilities provided by current RDBMS, most of such advanced techniques cannot be supported by databases. To our best knowledge, only the Pearson's correlation coefficient [37] can be implemented inside databases. Therefore, in this section, we extend the Pearson's correlation, namely inverted correlation, and provide corresponding database implementation.

4.1 Token Correlation

Assume that we have segmented an unstructured set record into a set of tokens, e.g. words or *q-grams*. The Pearson's correlation coefficient [37] can be used to compute the correlations between tokens t_i and t_j.

$$cor(t_i,t_j) = \frac{\Pr(t_i,t_j) - \Pr(t_i)\Pr(t_j)}{\sqrt{\Pr(t_i)\Pr(t_j)(1-\Pr(t_i))(1-\Pr(t_j))}} \quad (3)$$

where $\Pr(t_i)$ is the support of t_i, i.e. $\Pr(t_i) = \frac{f(t_i)}{N}$, $f(t_i)$ is the number of records containing token t_i and N is the total number of records. In other words, $\Pr(t_i)$ also denotes the probability that token t_i appears in a record in the database. However, according to the information property of texts, a token that appears frequently in the records in a dataset, i.e., high $f(t_i)$ value, might not be important in evaluating the correlations of records [33, 29]. For example, the word token "and" may appear frequently in set records. However, the correlation between token t_i and token "and" does not help greatly in finding correlated records.

Thus, to evaluate the correlations between tokens, we first introduce the concept of inverted probability which has been successfully adopted in the *inverse document frequency (idf)* [33, 29]. The *idf* is based on the essential intuition that a token appears frequently in different documents (records) is not a good discriminator and should be associated with a low feature weight; while a token with a low document frequency means that it is more relevant to those documents where it appears. The basic formula of *idf* is:

$$idf(t_i) = \log \frac{1}{\Pr(t_i)} = \log \frac{N}{f(t_i)} \quad (4)$$

where N denotes the total number of documents (records), and $f(t_i)$ is the number of documents (records) that contain token t_i. Similar to the case of documents in *idf*, we replace the documents by our set records in this study.

Specifically, motivated by successes of the inverted document frequency in retrieval of correlated text documents, we propose the inverted probability of a token t_i, which is

defined as:

$$Pi(t_i) = \log \frac{1}{\Pr(t_i)} = \log \frac{N}{f(t_i)} \quad (5)$$

where $Pi(t_i)$ denotes the inverted probability of token t_i. A higher frequency $f(t_i)$ of token t_i appearing in records indicates a lower inverted probability value $Pi(t_i)$. Moreover, considering the case that two tokens t_i and t_j appear together, we define the joint inverted probability of t_i and t_j as:

$$Pi(t_i, t_j) = \log \frac{1}{\Pr(t_i, t_j)} = \log \frac{N}{f(t_i, t_j)} \quad (6)$$

where $f(t_i, t_j)$ denotes the number of records where both token t_i and t_j appear, i.e. the joint frequency of (t_i, t_j).

Note that we can model the correlation relationship of token co-occurrence using a conditional probability. Given two tokens t_i and t_j, the conditional probability of t_j appearing in a record when t_i has already appeared in the same record is:

$$\Pr(t_j \mid t_i) = \frac{\Pr(t_i, t_j)}{\Pr(t_i)} \quad (7)$$

where $\Pr(t_i, t_j)$ denotes the probability of t_i and t_j appearing together in a record.

Now, similar to the conditional probability in Equation 7, we can compute the token correlations according to the inverted probability with token frequency. Note that the conditional probability describes an asymmetric relationship from t_i to t_j. With the inverted probability, we define the asymmetric correlation from token t_i to t_j as:

$$Cor(t_j \mid t_i) = \frac{Pi(t_i)}{Pi(t_i, t_j)} = \frac{\log \frac{N}{f(t_i)}}{\log \frac{N}{f(t_i, t_j)}} \quad (8)$$

where we have $Cor(t_j \mid t_i) \neq Cor(t_i \mid t_j)$. However, in the real world, we prefer to describing the correlations between two tokens in a symmetric way, i.e. if t_i is correlated with t_j then t_j is also correlated with t_i. Therefore, we define the token correlations formally in a symmetric style by considering both $Cor(t_j \mid t_i)$ and $Cor(t_i \mid t_j)$.

Definition 4.1 (Token Inverted Correlation). *Given two tokens t_i and t_j, the* token correlation *between t_i and t_j can be represented by:*

$$\begin{aligned} cor(t_i, t_j) &= Cor(t_j \mid t_i) Cor(t_i \mid t_j) \\ &= \begin{cases} \frac{\log \frac{N}{f(t_i)} \log \frac{N}{f(t_j)}}{(\log \frac{N}{f(t_i, t_j)})^2} & \text{if } f(t_i, t_j) \neq 0 \\ 0 & \text{if } f(t_i, t_j) = 0 \end{cases} \end{aligned} \quad (9)$$

where $f(t_i)$ and $f(t_j)$ denote the number of records containing t_i and t_j respectively, $f(t_i, t_j)$ is the number of records with both t_i and t_j, and N is the total number of records in the database.

The semantic meaning of the correlation between tokens t_i and t_j is described as the probabilistic relationship of these two tokens appearing together in the same records within the whole database. Specifically, the larger the value of $f(t_i, t_j)$ is, the higher the correlation between t_i and t_j. Thus, two tokens appearing together frequently have high correlation. On the other hand, for a fixed $f(t_i, t_j)$ observation, if the frequency values $f(t_i)$ and $f(t_j)$ of tokens t_i and t_j are large, the correlation is low. Thereby, if two tokens t_i and t_j always appear together in the same records, i.e. $f(t_i, t_j) = f(t_i) = f(t_j)$, we have the correlation $cor(t_i, t_j) = 1$. The intuition can also be extended to the correlation between the token t_i and itself, i.e. $cor(t_i, t_i) = 1$. If two tokens t_i and t_j never appear together in the same records, i.e. $f(t_i, t_j) = 0$, then there is no correlation between these two tokens, i.e., $cor(t_i, t_j) = 0$. In fact, according to the definition of token correlation, we have the following property:

Property 4.1. *If the correlation $cor(t_i, t_j)$ between two tokens t_i and t_j exists, we have the correlation value in the range of $0 < cor(t_i, t_j) \leq 1$.*

Proof. At first, in the real application, we have $f(t) \ll N$, i.e. $\frac{N}{f(t)} > 1$. If the correlation between two tokens t_i and t_j exists, then these two tokens should appear together in some records, i.e., $0 < f(t_i, t_j) \ll N$. Since each $\log \frac{N}{f(t)} > 0$ and $\log \frac{N}{f(t_i, t_j)} > 0$, the condition $0 < cor(t_i, t_j)$ is satisfied. Moreover, intuitively, t_i and t_j appearing together in the records implicates that t_i must exist in these records. Therefore, we have $f(t_i, t_j) \leq f(t_i)$ and $f(t_i, t_j) \leq f(t_j)$, which implies $\log \frac{N}{f(t_i)} \leq \log \frac{N}{f(t_i, t_j)}$, and

$$Cor(t_j \mid t_i) = \frac{\log \frac{N}{f(t_i)}}{\log \frac{N}{f(t_i, t_j)}} \leq 1 \quad (10)$$

It is the same for the $Cor(t_i \mid t_j) \leq 1$. Consequently, we also have $cor(t_i, t_j) \leq 1$. To sum up the above arguments, the correlation with $0 < cor(t_i, t_j) \leq 1$ is satisfied. □

4.2 Database Implementation

We start from the dataset that has already been represented in the table $records(tid, rid)$. First, we count the times of each token pair appearing in the records, i.e., $f(t_i, t_j)$ in equation 9, and store the generated tuples $(t_i, t_j, f(t_i, t_j))$ in the correlation view with schema $correlation(tid_1, tid_2, weight)$. Note that $f(t_i)$ of each token is also computed by this step and represented as $(t_i, t_i, f(t_i))$. Then, we count all the number of records in the table of *records*, namely `@N`. Next, we perform the correlation computation according to Equation 9. Consider a tuple of correlation `c3` in the algorithm with `(tid1, tid2, weight)`. Let `c1.weight` be the frequency of token `c3.tid1` and `c2.weight` be the frequency of token `c3.tid2`, according to the definition in Equation 9 the correlation weight of `c3` can be computed as:

$$\frac{\texttt{LOG(@N/c1.weight)} * \texttt{LOG(@N/c2.weight)}}{\texttt{POWER(LOG(@N/c3.weight), 2)}} \quad (11)$$

Finally, we update the correlation weight in *correlation*. The SQL statement for token correlation construction is described as follows.

```
INSERT INTO correlation
 SELECT DISTINCT r1.tid, r2.tid, COUNT(*)
 FROM records AS r1, records AS r2
 WHERE r1.rid = r2.rid
 GROUP BY r1.tid, r2.tid

DECLARE @N INT
SET @N = (SELECT COUNT(DISTINCT rid)
  FROM records)
```

```
UPDATE correlation
SET    weight = temp.w
FROM
 (SELECT DISTINCT
   (LOG(@N/c1.weight) * LOG(@N/c2.weight) /
    POWER(LOG(@N/c3.weight),2)
   )AS w, c3.tid1 AS id1, c3.tid2 AS id2
  FROM correlation c1, correlation c2,
       correlation c3
  WHERE c1.tid1=c3.tid1 AND c1.tid2=c3.tid1
    AND c2.tid1=c3.tid2 AND c2.tid2=c3.tid2
 ) AS temp
WHERE tid1 = temp.id1 AND tid2 = temp.id2
```

5. CORRELATION FILTER

According to the correlation query framework, the number of token correlations affects performance of correlation queries. Thus, we try to improve the query efficiency by reducing the size of token correlations. Specifically, we investigate the interaction between the number of token correlations (correlation count) and the importance of token correlations (correlation weight). Intuitively, two tokens with low correlation weights have small probability of appearing together, in other words, they are less important in our correlation query. In fact, noisy data often exist in real applications, and some false correlations with low weights might be generated by isolated noise. Therefore, we can set a minimum correlation threshold η to filter out those token correlations with low correlation weights. If the threshold η is small, noisy correlations are taken into account in ranking the results. Consequently, the noisy correlations affect the results largely. On the other hand, with the increase of threshold η, only those correlations with high weights are reserved, which are close to 1. Thereby, the difference between the correlation query and similarity matching query also becomes small, and two approaches tend to get similar query results. From the above discussion, we can find out that setting a proper filtering threshold is essential to improve both the efficiency and effectiveness of a correlation query.

5.1 Foundations

A good filter should remove non-important correlations and reserve the important correlations as many as possible. Here, the importance of correlations is represented by the correlation weight. In the rest of this section, we provide an estimated range of correlations as a guideline of threshold selection, and prove that we can use less number of correlations to represent more important correlations in that correlation range.

In order to evaluate the balance of reserving more important correlation information and reducing more number of correlations, we compare the distributions of correlations on correlation numbers and correlation weights. We show the statistics of correlation distributions of *RCV1* and *NSF* datasets in Figure 3. For a correlation weight x, the frequency distribution bar $f(x)$ denotes the normalized number of correlations with the weight x, i.e., correlation weight distributions. From Figure 3, we find that the correlations are not distributed uniformly in the whole range of correlation value $(0, 1]$. A curve of the *nominal distribution* is also plotted to compare with the frequency distribution, and we observe similar patterns in both two datasets.

Based on the frequency distribution observation on the *RCV1* and *NSF* datasets, we model the correlation distribution as the *nominal distribution*. Let random variable X be the token correlation weight, we consider the following two distributions.

DISTRIBUTION 5.1. *$N_c(\mu_c, \sigma_c^2)$: The distribution of X in terms of the number of correlations. Each correlation is treated as a unit of statistic sample. The* probability density function *$f_c(x)$ describes the probability distribution of X on the number of correlations. The* cumulative distribution function *$F_c(x)$ denotes the probability of X less than x, $P(X \leq x)$, in terms of correlation count. The estimation of μ_c and σ_c^2 can be described as:*

$$\hat{\mu}_c = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{12}$$

$$\hat{\sigma}_c^2 = \frac{1}{n}\sum_{i=1}^{n} (x_i - \bar{x})^2 \tag{13}$$

where n is the total number of correlations. The probability density function *of N_c represents the distribution of X on correlation counts, e.g. how many number of correlations with the correlation weight in the range of $0.1 < X < 0.2$.*

DISTRIBUTION 5.2. *$N_s(\mu_s, \sigma_s^2)$: The distribution of X in terms of the weight of correlations. Each unit of correlation weight is treated as a statistic sample, while each correlation with weight x_i is treated as a number of x_i statistic samples. The* probability density function *$f_s(x)$ describes the probability distribution of X on the weight of correlations. The* cumulative distribution function *$F_s(x)$ denotes the probability of X less than x, $P(X \leq x)$, in terms of correlation weight. The estimation of μ_s and σ_s^2 is given by*

$$\hat{\mu}_s = \bar{x} = \frac{1}{\sum_{i=1}^{n} x_i}\sum_{i=1}^{n} x_i x_i \tag{14}$$

$$\hat{\sigma}_s^2 = \frac{1}{\sum_{i=1}^{n} x_i}\sum_{i=1}^{n} x_i (x_i - \bar{x})^2 \tag{15}$$

where $\sum_{i=1}^{n} x_i$ denotes the total units of correlation weights. The variable X is the same with the distribution 1, but the distributions of the variable are different. In the distribution 1, one correlation with weight x_i is counted as 1 time, however, for distribution 2, we treat the correlation as x_i units and count one correlation with weight x_i for x_i times. The probability density function of distribution 2 represents the distribution of X on correlation weights, e.g. how many weight units of correlations with the correlation weights in the range of $0.1<X<0.2$. Recall that our record correlation function also aggregates the correlation weights.

Now we discuss the selection of the minimum threshold η, in order to efficiently and effectively achieve the optimal query results. First, we present the relationship between the estimate values μ_c and μ_s of distribution 1 N_c and distribution 2 N_s. Then, for a given value x between μ_c and μ_s, we can exploit the bounds of the cumulative distribution function $F_c(x)$ and $F_s(x)$. Finally, we prove that by setting a certain filtering threshold, we can remove more less important correlations without losing much query effectiveness.

THEOREM 5.1. *Let μ_c be the estimate value of distribution 1 N_c and μ_s be the estimate value of distribution 2 N_s, and then we have $\mu_c \leq \mu_s$.*

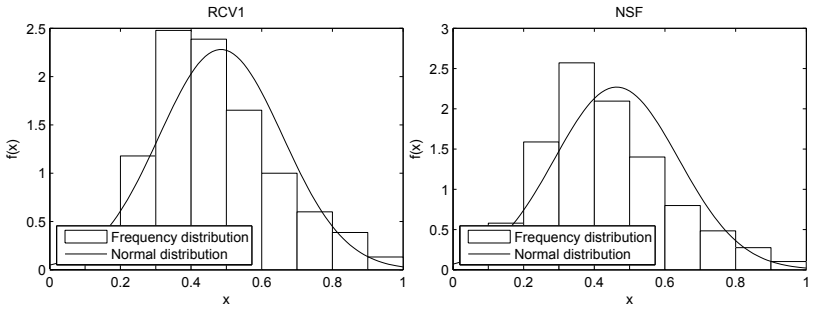

Figure 3: Frequency distribution of correlations

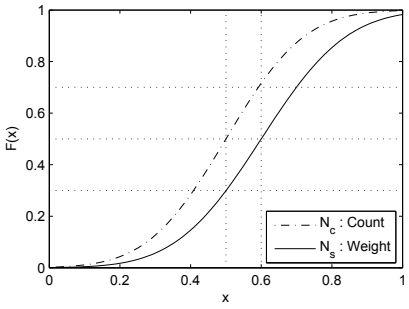

Figure 4: Cumulative distribution

PROOF. According to the definition of correlation, all the correlation weights have $0 < x_i < 1$. Therefore,

$$\begin{aligned}\mu_s - \mu_c &= \frac{1}{\sum_{i=1}^{n} x_i}\sum_{i=1}^{n} x_i x_i - \frac{1}{n}\sum_{i=1}^{n} x_i \\ &= \frac{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}{n\sum_{i=1}^{n} x_i} \\ &= \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i^2 + x_j^2) - \sum_{i=1}^{n}\sum_{j=1}^{n} 2x_i x_j}{2n\sum_{i=1}^{n} x_i} \\ &= \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i - x_j)^2}{2n\sum_{i=1}^{n} x_i} \geq 0\end{aligned}$$

i.e. we have proved $\mu_c \leq \mu_s$. □

THEOREM 5.2. *Let $F_c(x)$ be the cumulative distribution function of distribution N_c, $F_s(x)$ be the cumulative distribution function of distribution N_s. If the value $x \in [\mu_c, \mu_s]$, and then we have $F_s(x) \leq 0.5 \leq F_c(x)$*

PROOF. If a variable X is normally distributed with mean μ and variance σ^2, the definition of *cumulative distribution function* $F(x)$ can be described as:

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt \tag{16}$$

According to the property of cumulative distribution function, we have $F(\mu) = 0.5$; for $x \geq \mu$, then $F(x) \geq 0.5$; and vice versa. Consider the value $x \in [\mu_c, \mu_s]$, in the distribution N_c, we have $x \geq \mu_c$ thus $F_c(x) \geq 0.5$; and also in the distribution N_s, $x \leq \mu_s$ indicates $F_s(x) \leq 0.5$. □

THEOREM 5.3. *By setting the minimum threshold η of correlation weight in the range of $\mu_c \leq \eta \leq \mu_s$, we can use less than 50% number of correlations to represent more than 50% correlation weights (importance).*

PROOF. Recall that $F_c(x)$ denotes the proportion of the number of correlations with weights less than x, while $F_s(x)$ denotes the proportion of the total weights (importance) of correlations with weights less than x. For a certain threshold η, the value of cumulative distribution function $F(\eta)$ indicates the proportion that are filtered out and $(1 - F(\eta))$ represents the reserved parts. According to THEOREM 5.2, by setting the minimum threshold η of correlation weights in the range of $\mu_c \leq \eta \leq \mu_s$, the lower bound of the number of correlations that are filtered out is 50%; and the upper bound of the total weight of correlations that are filtered out is also 50%. □

Therefore, by setting a minimum correlation threshold $\eta \in [\mu_c, \mu_s]$, we can remove more than 50% of less important correlations, but lose less than 50% correlation weights. In Figure 4, we present an example of the cumulative distribution function $F(x)$. The μ_c of N_c is about 0.5 and the μ_s of N_s is about 0.6. In the range of $x \in [0.5, 0.6]$, we have $F_c(x) \geq 50\%$ and $F_s(x) \leq 50\%$. With a threshold of η ($0.5 \leq \eta \leq 0.6$), for example $\eta = 0.5$, we can filter out about 50% of correlations but only lose 30% of total correlation weights. In other words, by the filter, we reduce the size of correlations largely but with little loss on the correlation weights.

So far, we have proved that by using a filter with a threshold η around the range of $[\mu_c, \mu_s]$, we can remove more correlations (counts) and lose less correlation importance (weights). In practice, given a set records database, we can compute token correlation distribution first, and then set up the filtering threshold according to the above derived optimal threshold range.

5.2 Correlation Filter Implementation

The following algorithm estimates the parameters of μ_c and μ_s. The first query returns the estimation of μ_c according to the definition in equation 12, and the second query result is the estimation of μ_s according to the definition in equation 14. Note that the correlation weight between token t_i to itself is constant, i.e. $cor(t_i, t_i) = 1$. Therefore, we do not take those tuples of the self-correlations into account and set a condition of `tid1 <> tid2` in the algorithm. Finally, we can implement the filter with threshold η and remove all the correlations with weights less than the specified threshold η.

```
SELECT SUM(weight) / COUNT(*)
FROM correlation
WHERE tid1 <> tid2

SELECT SUM(weight*weight) / SUM(weight)
FROM correlation
WHERE tid1 <> tid2

DELETE FROM correlation
WHERE weight < $\eta$
```

6. EXPERIMENTAL EVALUATION

We now present our extensive experimental evaluation, in terms of both effectiveness and efficiency. We compare the correlation query with the similarity matching query. Since the edit operation based approaches like *edit distance*

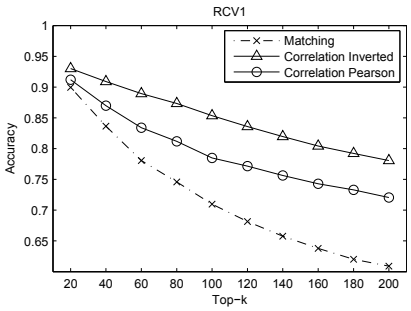

Figure 5: Effectiveness on *RCV1*

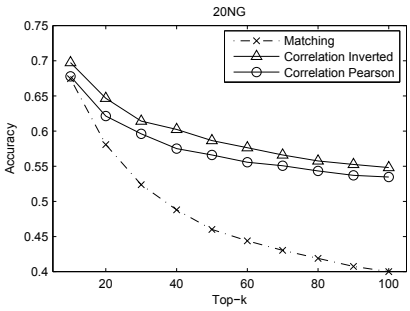

Figure 6: Effectiveness on *20NG*

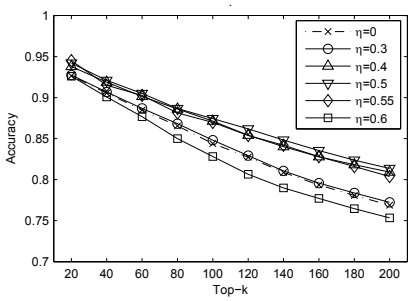

Figure 7: Filter evaluation

can only capture limited similarity and fail in many cases such as various word orders [14], we adopt the vector space based approach as the similarity matching technique, i.e., the matching coefficient $sim(q, r) = |q \cap r|$ of text [31]. We also compare our inverted correlation in formula 9 with the Pearson's correlation coefficient [37] in formula 3.

6.1 Experimental Settings

The experiments are conducted on a machine with Intel Core 2 CPU (2.13 GHz) and 2 GB of memory. RDBMS uses SQL Server 2005.

Datasets. We run the experiments on *Reuters Corpus Volume I (RCV1)* [23], which is also a widely used benchmark dataset with category labels in text retrieval area. We abstract m ($m = 10$ in this experiment) words with the highest *tf*idf* value from each article as the summary, and merge these m words to an unstructured set record. The category label of each article is also utilized as the label of corresponding record. The second dataset we used in the effectiveness experiments is the *20 Newsgroups (20NG)* [21]. The *20NG* data are also text entries with data category labels. Similar to the *RCV1*, we use the *20NG* dataset to evaluate the effectiveness in the same way[1].

Evaluation Criteria. The evaluation criteria address two aspects in the experiments, i.e. the effectiveness and the efficiency. Specifically, we select 100 records from the dataset as queries. For the time performance evaluation, we conduct all the queries and compute the response time as runtime costs. In addition, we also report the number of token correlations left after a minimum correlation threshold applied, since it is directly related to the running time. For the effectiveness, we evaluate the accuracy in the top-k query results.

$$accuracy = \frac{\text{number of results with the same label of } q}{\text{total number of results } k} \tag{17}$$

The *accuracy* value denotes the correctness of query results, i.e., the query and the results belong to the same labeled category. Higher *accuracy* of a query means better effectiveness. Here, we do not adopt the *f-measure* [35] with recall to evaluate the completeness of the top-k results. It is meaningful to tell the high correlation between "NBA" and "basketball", but unreasonable (in real applications) to achieve a complete answer set in top-k results, i.e., to enumerate and return all possible records that are correlated with the query "basketball". Therefore, rather than evaluating the recall, we mainly concern the accuracy of correlation query.

[1]Similar results on *20NG* are not reported in some experiments, due to the limitation of space.

6.2 Evaluation on Effectiveness

The first experiment is focused on the effectiveness comparison of correlation query and similarity matching-based query. In this experiment, we do not conduct the filter on token correlations. The program runs on 50,000 records in the $RCV1$ dataset and 10,000 records in the $20NG$ dataset. Figure 5 and 6 shows the accuracy of top-k results by three different approaches, including the similarity matching, the inverted correlation proposed in this paper, and the Pearson's correlation coefficient. As shown in the results, these three approaches achieve high accuracy in the top-20 results. These highly correlated result records have considerable common tokens with each other, thus the similarity matching-based query can also identify the correlation relationship. However, when larger sizes of query results are needed, for example top-100 or top-200, the relationships between the query and records can hardly be detected by only using the overlaps of matching tokens. Therefore, with the increase of result sizes, the accuracy of similarity matching-based query drops quickly, while our correlation query achieves comparatively high accuracy. Most importantly, our token inverted correlation shows a higher accuracy comparing with the Pearson's correlation coefficient, which also verifies the derivation in Section 4.1. To summarize, the experiment results demonstrate the effectiveness of our correlation queries.

6.3 Evaluation on Correlation Filter

In this experiment, we study the improvement of effectiveness by applying the correlation filter. Recall that noisy correlations may affect the query results. The filtering strategy is to set a minimum correlation weight threshold η. We evaluate the effect of correlation filter by reporting the results of accuracy (Figure 7).

First, we can improve the effectiveness by introducing a filter in the correlation query. As shown in the Figure 7, the accuracy of query results improves by using the minimum threshold filter $\eta = 0.4$. By setting this threshold, we filter out the correlations with weights less than 0.4, which are less important (such as noisy data). On the other hand, if the threshold is set too large, the accuracy drops as shown in Figure 7 with $\eta = 0.6$. As we increase the threshold η, more correlations are filtered out, and the query can only rely on the matching tokens and few correlations still left. Thus, the results of correlation query are closer to the similarity matching ones, when the threshold is brought near 1.

To estimate the filtering threshold, we observe the distribution of correlations at first, and compute the estimation value of $\mu_c = 0.505$ and $\mu_s = 0.566$ according to the definition in Section 4.2. According to Lemma 5.3, by set-

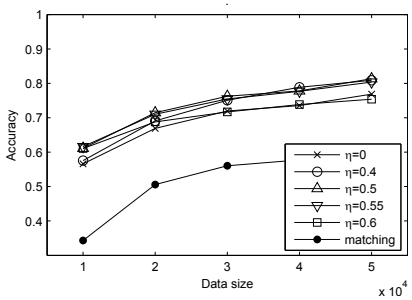

Figure 8: Scalability on effectiveness

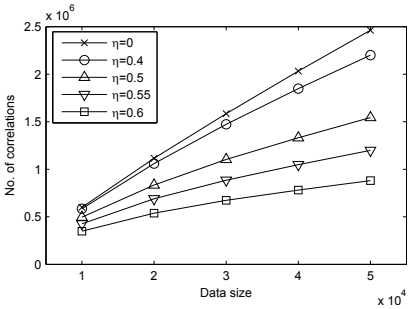

Figure 9: Scalability on filter

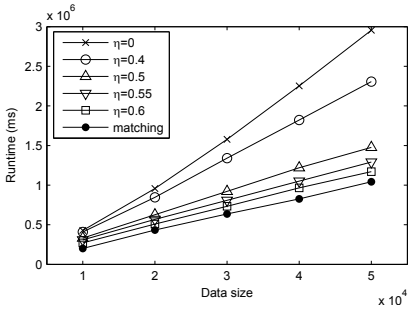

Figure 10: Scalability on efficiency

ting $\mu_c \leq \eta \leq \mu_s$, we can reduce more than 50% number of correlations while reserve more than 50% correlation information. Therefore, in order to improve the efficiency without losing too much accuracy, the threshold η is preferred around the range of $[\mu_c, \mu_s]$ to guarantee the tradeoff between the correlation count (size) and correlation weight (importance). The results in Figure 9 also confirm that by setting $\eta = 0.55 \in [0.505, 0.566]$, we can reduce the number of correlations significantly and with a little loss on effectiveness (Figure 8) compared with $\eta = 0.4$ or 0.5.

In fact, the estimation of μ_c and μ_s is a guideline strategy of choosing threshold. As the threshold increases, the values of $F_c(x)$ and $F_s(x)$ become closer in the cumulative distribution function. In other words, the difference between the number of correlations (N_c) and the importance of correlations (N_s) is smaller. Since we could no longer keep all highly important correlations by a smaller number of correlations, the accuracy drops quickly as shown in Figure 7 with $\eta = 0.6$.

6.4 Evaluation on Scalability

This experiment demonstrates the scalability of our approach in different data sizes. We perform a number of 100 queries on the data sizes from 10,000 to 50,000 respectively. For the effectiveness, we observe the number of results returned in different data scales and the accuracy of the top-200 results for each data size. For the efficiency, we study the correlation size under several filters and compare the runtime costs with the similarity matching-based query.

In Figure 8, we compare the effectiveness of different approaches. Our correlation query achieves a higher accuracy under different data sizes. Moreover, the results of correlation query with different filters are quite similar. By increasing the threshold, the accuracy arises at $\eta = 0.4$ and $\eta = 0.5$, then drops back to the original level when $\eta = 0.6$, but still without losing too much accuracy.

In Figure 10, we illustrate the time performance of our approach. The runtime costs largely depend on the size of correlations in the computation. As shown in Figure 9, the number of correlations increases approximately in linear with the increasing of dataset size under different thresholds. By applying a filter, we can reduce the runtime costs largely. When the threshold $\eta = 0.6$, the time costs of correlation query are even quite similar to the similarity matching-based approach, under all of the data sizes. Meanwhile, the effectiveness of correlation query (Figure 8) is much higher than the similarity matching query.

We also evaluate the scalability on the *NSF* dataset. Since the records in the *NSF* dataset are not labeled with categories, we only conduct the experiments in terms of time performance. The results are presented in Figure 11. We get similar curves of time costs as the results shown in Figure 8 on *RCV1* dataset. The time costs increase linearly as the increase of data size. When the threshold is set to be $\eta = 0.6$, the time costs of correlation query are comparable to the similarity matching-based query. On the other hand, as the same results as described in the previous experiments, the correlation query is more effective than the similarity matching approach.

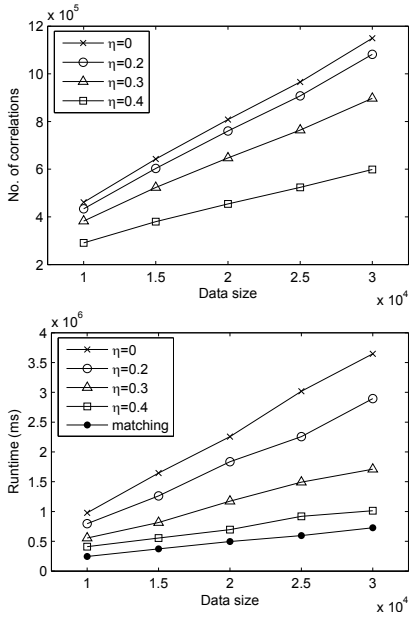

Figure 11: Scalability on efficiency of *NSF*

7. CONCLUSIONS

In this paper, motivated by the significance of supporting correlation query over set record in databases, we propose a novel query framework for set correlation query by using RDBMS facilities. To our best knowledge, this is the first work on supporting set correlation query inside databases, and only the Pearson's correlation can be implemented to construct token correlations by using current RDBMS facilities. Thereby, we study a novel correlation coefficient to extend Pearson's correlation, and provide a pure-SQL implementation inside databases. We emphasize this pure-SQL implementation of set correlation query inside RDBMS, which can be naturally extended to set correlation join over set records as well. Moreover, our theoretical analysis proves that, with a proper setting of filtering on token correlations, we can improve the query efficiency with a little effectiveness

loss. Finally, our experiments demonstrate the superiority of our approach in terms of both effectiveness and efficiency.

Acknowledgments

This work is supported by HONG KONG RGC GRF grant under project no. 611608.

8. REFERENCES

[1] J. Allan, C. Wade, and A. Bolivar. Retrieval and novelty detection at the sentence level. In *SIGIR*, pages 314–321, 2003.

[2] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *VLDB*, pages 918–929, 2006.

[3] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *VLDB*, pages 918–929, 2006.

[4] N. Balasubramanian, J. Allan, and W. B. Croft. A comparison of sentence retrieval techniques. In *SIGIR*, pages 813–814, 2007.

[5] J. L. Beckmann, A. Halverson, R. Krishnamurthy, and J. F. Naughton. Extending RDBMSs to support sparse datasets using an interpreted attribute storage format. In *ICDE*, page 58, 2006.

[6] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD*, pages 39–48, 2003.

[7] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *SIGMOD Conference*, pages 265–276, 1997.

[8] G. Cao, J.-Y. Nie, and J. Bai. Integrating word relationships into language models. In *SIGIR*, pages 298–305, 2005.

[9] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, page 5, 2006.

[10] P.-A. Chirita, C. S. Firan, and W. Nejdl. Personalized query expansion for the web. In *SIGIR*, pages 7–14, 2007.

[11] W. W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *SIGMOD Conference*, pages 201–212, 1998.

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

[13] P. Fung and L. Y. Yee. An ir approach for translating new words from nonparallel, comparable texts. In *COLING-ACL*, pages 414–420, 1998.

[14] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an rdbms for web data integration. In *WWW*, pages 90–101, 2003.

[15] M. Hadjieleftheriou, X. Yu, N. Koudas, and D. Srivastava. Hashed samples: selectivity estimators for set similarity selection queries. *PVLDB*, 1(1):201–212, 2008.

[16] T. Hofmann. Probabilistic latent semantic analysis. In *UAI*, pages 289–296, 1999.

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

[18] A. Jain, A. Doan, and L. Gravano. SQL queries over unstructured text databases. In *ICDE*, 2007.

[19] C. Jermaine. The computational complexity of high dimensional correlation search. In *ICDM*, pages 249–256, 2001.

[20] R. Jin, J. Y. Chai, and L. Si. Learn to weight terms in information retrieval using category information. In *ICML'05*, pages 353–360, 2005.

[21] K. Lang. Newsweeder: Learning to filter netnews. In *ICML*, pages 331–339, 1995.

[22] H. Lee, R. T. Ng, and K. Shim. Power-law based estimation of set similarity join size. *PVLDB*, 2(1):658–669, 2009.

[23] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[24] X. Li and W. B. Croft. Novelty detection based on sentence level patterns. In *CIKM*, pages 744–751, 2005.

[25] X. Li and W. B. Croft. Improving novelty detection for general topics using sentence level information patterns. In *CIKM*, pages 238–247, 2006.

[26] S. Liu, F. Liu, C. Yu, and W. Meng. An effective approach to document retrieval via utilizing wordnet and recognizing phrases. In *SIGIR*, pages 266–272, 2004.

[27] D. Metzler, S. T. Dumais, and C. Meek. Similarity measures for short segments of text. In *ECIR*, pages 16–27, 2007.

[28] V. Murdock and W. B. Croft. A translation model for sentence retrieval. In *HLT/EMNLP*, 2005.

[29] S. Robertson. Understanding inverse document frequency: On theoretical argument for idf. *Journal of Documentation*, 60(5):503–520, 2004.

[30] M. Sahami and T. D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *WWW*, pages 377–386, 2006.

[31] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley, 1989.

[32] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *KDD*, pages 269–278, 2002.

[33] K. Sparck Jones. Index term weighting. *Information Storage and Retrieval*, 9(11):619–633, 1973.

[34] M. Theobald, R. Schenkel, and G. Weikum. Efficient and self-tuning incremental query expansion for top-k query processing. In *SIGIR*, pages 242–249, 2005.

[35] C. J. Van Rijsbergen. *Information Retrieval.* Butterworth-Heinemann, Newton, MA, USA, 1979.

[36] R. W. White and J. M. Jose. A study of topic similarity measures. In *SIGIR*, pages 520–521, 2004.

[37] H. Xiong, S. Shekhar, P.-N. Tan, and V. Kumar. Exploiting a support-based upper bound of pearson's correlation coefficient for efficiently identifying strongly correlated pairs. In *KDD*, pages 334–343, 2004.

[38] H. Xiong, S. Shekhar, P.-N. Tan, and V. Kumar. Taper: A two-step approach for all-strong-pairs correlation query in large databases. *IEEE Trans. Knowl. Data Eng.*, 18(4):493–508, 2006.