World Wide Web (2010) 13:497–522
DOI 10.1007/s11280-010-0094-0

Mining discriminative items in multiple data streams

Zhenhua Lin · Bin Jiang · Jian Pei · Daxin Jiang

Received: 23 January 2010 / Revised: 11 June 2010 /
Accepted: 15 June 2010 / Published online: 10 July 2010

Abstract How can we maintain a dynamic profile capturing a user's reading interest against the common interest? What are the queries that have been asked 1,000 times more frequently to a search engine from users in Asia than in North America? What are the keywords (or tags) that are 1,000 times more frequent in the blog stream on computer games than in the blog stream on Hollywood movies? To answer such interesting questions, we need to find discriminative items in multiple data streams. Each data source, such as Web search queries in a region and blog postings on a topic, can be modeled as a data stream due to the fast growing volume of the source. Motivated by the extensive applications, in this paper, we study the problem of mining discriminative items in multiple data streams. We show that, to exactly find all discriminative items in stream S_1 against stream S_2 by one scan, the space lower bound is $\Omega(|\Sigma| \log \frac{n_1}{|\Sigma|})$, where Σ is the alphabet of items and n_1 is the current size of S_1. To tackle the space challenge, we develop three heuristic algorithms that can achieve high precision and recall using sub-linear space and sub-linear processing time per item with respect to $|\Sigma|$. The complexity of all algorithms are independent

The authors are grateful to the anonymous reviewers for their constructive and insightful comments on the paper. This research was supported in part by an NSERC Discovery Grant and an NSERC Discovery Accelerator Supplement Grant. All opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

Z. Lin · B. Jiang · J. Pei (✉)
Simon Fraser University, 8888 University Drive, Burnaby, Canada
e-mail: jpei@cs.sfu.ca

Z. Lin
e-mail: ulysses_lin@cs.sfu.ca

B. Jiang
e-mail: bjiang@cs.sfu.ca

D. Jiang
Microsoft Research Asia, 49 Zhichun Road, Beijing, China
e-mail: djiang@microsoft.com

to the size of the two streams. An extensive empirical study using both real data sets and synthetic data sets verifies our design.

Keywords data mining · data streams · discriminative items

1 Introduction

We want to build a personalized news delivery service. When a user joins the system, we have no idea about the user's profile, and thus we start to provide all news topics to the user. As the user keeps reading some news articles, how can we maintain a dynamic profile capturing the user's reading interest? One meaningful approach is to find the keywords that are much, say, 1,000 times, more frequent in the articles read by the user than in the collection of all articles. We can use the profile to search the news articles in the future to achieve a dynamic personalized service. However, this problem is far from trivial since the user's reading interest is dynamic and may change from time to time. Moreover, news articles as well as the articles read by the user keep arriving as data streams.

Problems carrying the similar nature can be found in many aspects of Web search. For example, a search engine may want to monitor the search queries that are asked 1,000 times more frequently in a region, say Asia, than in another region, say North America. Such queries are very useful for the search engine in query optimization, localization, and suggestion. As another example, tagging and blogging are common exercises on the Web now. One may wonder, comparing to the blog postings on Hollywood movies, which tags are 1,000 times more frequent in the blog postings on computer games. Those tags provide a means to characterize the ongoing topic of computer games and the differences from Hollywood movies. Such information is also useful in analyzing a social network of bloggers.

If one wishes, the list of similar examples can easily continue. For example, one may compare the tags on images taken by different user groups to understand the users' interest. Moreover, in Intranet, one may compare activities and documents in failed projects against those in successful projects to obtain hints of problems in projects. To name one more, it is interesting to monitor the advertisements that are clicked much more frequently by mobile users than those by other users so that we can understand the differences in user preferences for sponsored search.

The above examples motivate a problem of mining discriminative items in data streams. Due to the large and fast growing volumes of those data sources such as Web search queries in a region and blog postings and tags on a topic, each data source can be modeled as a data stream, for which only one scan of data is allowed by the computation resource or application requirements. We want to compare two data streams S_1 and S_2, and maintain the collection of items such that their frequencies in S_1 are θ times more than their frequencies in S_2, where θ is a user specified parameter.

The problem of mining discriminative items in data streams is also related to the conditional topic model [4, 5, 33]. A topic can be modeled as a keyword distribution describing the topic. Then, a conditional topic model of "computer games" against "Holleywood movies" is the distribution of keywords in the documents related to "computer games" conditional on the distribution of keywords in the documents related to “Holleywood movies”. The discriminative keywords can be regarded as the points of high density in the conditional distribution.

Although finding frequent items in a single stream is well studied (see Section 7 for a brief review), little work has been done to find discriminative items over multiple streams, mainly due to the difficulty of finding infrequent items in a stream [19].

In this paper, we tackle the problem of mining discriminative items on multiple data streams. We make the following contributions. First, we show that, to exactly find all discriminative items in stream S_1 against stream S_2 by one scan, the space lower bound is $\Omega(|\Sigma| \log \frac{n_1}{|\Sigma|})$, where Σ is the alphabet of items and n_1 is the current size of S_1. The lower bound clearly indicates that any exact one-scan method for mining discriminative items is infeasible for online applications since a stream grows constantly in size and the alphabet such as tags and queries often grows fast, too. To tackle the space challenge, we develop three heuristic algorithms that can achieve high precision and recall using sub-linear space and sub-linear processing time per item with respect to $|\Sigma|$. The complexity of all algorithms are independent of the size of the two streams. We report an extensive empirical study using both real data sets and synthetic data sets to verify our design.

The rest of the paper is organized as follows. In Section 2, we formulate the problem of mining discriminative items over data streams and give a space lower bound. In Section 3, we develop a frequent item based method which derives discriminative items from frequent items in a single stream. We devise a hash-based method in Section 4. In Section 5, we integrate the advantages of the frequent item based method and the hash-based method, which consumes the least space to achieve high precision and recall. Section 6 reports extensive experiments on real and synthetic data sets and shows that our methods are efficient and scalable. Section 7 reviews the related work. Section 8 concludes the paper.

2 Problem definition

In this section, we first formulate the problem of mining discriminative items from streams. Then, we give a space lower bound. Last, we summarize the theoretical results.

2.1 Discriminative items in data streams

Given an alphabet of items Σ, we consider two streams S_1 and S_2 which are composed of occurrences of items in Σ. Denote by n_1 and n_2 the current sizes of S_1 and S_2, respectively. We do not require that two streams are synchronized.

Let $f_i(e)$ $(i = 1, 2)$ denote the *frequency*, or the number of occurrences, of an item e in S_i. We also define the *frequency rate* of e in S_i $(i = 1, 2)$ as $r_i(e) = \frac{f_i(e)}{n_i}$.

We are interested in discriminative items which are relatively frequent in S_1 but relatively infrequent in S_2. Formally, an item e is a *discriminative item* if

$$R(e) = \frac{r_1(e)}{r_2(e)} = \frac{f_1(e)n_2}{f_2(e)n_1} \geq \theta,$$

where $\theta > 1$ is a user specified threshold. The larger the value of θ, the more discriminative the item. In many applications, we favor a large θ, such as in the order of hundreds or thousands.

To deal with the cases where $f_2(e) = 0$, We introduce a user specified threshold $0 < \phi < \frac{1}{\theta}$, and require that any discriminative item should have a frequency in S_1 no less than $\phi\theta n_1$. ϕ is called the minimum support threshold in S_1. The rationale is that infrequent items are not of significance in many applications. For example, a query seldom asked is not very interesting to a search engine. By this remedy, when an item e is not observed in S_2 (i.e., $f_2(e) = 0$), whether e is discriminative or not is determined by the condition $f_1(e) \geq \phi\theta n_1$.

2.2 Space lower bound

Let E denote the set of discriminative items, we establish the fact that any one-scan algorithm that can compute the exact E must use $\Omega(|\Sigma| \log \frac{n_1}{|\Sigma|})$ space in the worst case.

Theorem 1 (Space lower bound) *Any one-scan algorithm that computes the exact set of discriminative items E requires $\Omega(|\Sigma| \log \frac{n_1}{|\Sigma|})$ space in the worst case, where $|\Sigma|$ is the size of the alphabet Σ.*

Proof We reduce the problem of computing the exact set of frequent items to the problem of mining discriminative items. Given a stream S whose current size is n, let us consider finding all items in S with a minimum frequency αn where $\frac{1}{|\Sigma|} < \alpha < 1$. We construct a stream S' such that S' contains $\frac{1}{\phi}$ ($\phi < \alpha$) distinct items each of which appears once. This can be done using $\frac{1}{\phi}$ space which is less than the complexity stated in the theorem. We also set $\theta = \frac{\alpha}{\phi}$. Then, an item e is a discriminative item in S against S' with ratio θ if and only if e has a frequency αn in S. Therefore, any exact algorithm computing the set of discriminative items between two streams can be used to find the exact set of frequent items from one stream.

Karp et al. [22] (Proposition 2.1) showed that that any online algorithm that can find the exact set of frequent items, whose frequencies are no less than αn, requires $\Omega(|\Sigma| \log \frac{n}{|\Sigma|})$ space in the worst case. Thus, any one-scan algorithm that can compute the exact set of discriminative items E must use $\Omega(|\Sigma| \log \frac{n_1}{|\Sigma|})$ space in the worst case. □

Given two streams for mining discriminative items, $\frac{n_1}{n_2}$ is fixed for all items and can be treated as a constant. Without loss of generality, in the rest of the paper, we assume $n_1 = n_2 = n$ to keep our discussion simple. Consequently, we use a simplified definition of the discriminative item as follows.

Definition 1 (Discriminative items) Given an alphabet of items Σ, two streams S_1 and S_2, whose current sizes are n, a minimum ratio parameter $\theta > 1$, and a minimum support threshold $\phi \in (0, \frac{1}{\theta})$, an item e is *discriminative* in S_1 against S_2 if $e \in \Sigma$, $f_1(e) \geq \phi\theta n$ and $R(e) = \frac{f_1(e)}{f_2(e)} \geq \theta$. The problem of *mining discriminative items* in S_1 against S_2 is to find the set of discriminative items

$$E = \{e \in \Sigma | f_1(e) \geq \phi\theta n \wedge R(e) \geq \theta\}.$$

We note that, when $n_1 \neq n_2$, we simply multiply the simplified $R(e)$ with a constant $\frac{n_2}{n_1}$. The algorithms, proofs, and complexities presented in the rest of the paper can be extended in the same way in the cases where $n_1 \neq n_2$.

Example 1 (*Discriminative items*) Table 1 shows our running example. The alphabet $\Sigma = \{x, y, z, w\}$. Two streams S_1 and S_2 are of size 10 each. Items are shown from left to right in the table in the arriving order. The frequencies of x, y, z, and w in S_1 are 4, 2, 3, and 1, respectively, and in S_2 1, 4, 1, and 4, respectively. Let $\theta = 3$ and $\phi = 0.1$. Then, x and z are the discriminative items.

Next, we give an upper bound on the number of discriminative items.

Theorem 2 (The number of discriminative items) *Given two streams S_1 and S_2, a ratio threshold θ, and a minimum support threshold ϕ, there are at most* $\min\{|\Sigma|, \frac{1}{\phi\theta}\}$ *discriminative items.*

Proof It is trivial that $|E| \leq |\Sigma|$. We prove $|E| \leq \frac{1}{\phi\theta}$ by contradiction. Suppose $|E| > \frac{1}{\phi\theta}$. Because for any $e \in E$, $f_1(e) \geq \phi\theta n$, we have

$$\sum_{e \in E} f_1(e) \geq |E|\phi\theta n > n.$$

This contradicts that the current size of stream S_1 is n. Thus, $|E| \leq \frac{1}{\phi\theta}$, and $|E| \leq \min\{|\Sigma|, \frac{1}{\phi\theta}\}$. □

2.3 Summary of our heuristic methods

The lower bound clearly indicates that any exact one-scan method for mining discriminative items is infeasible for online applications since a stream grows constantly in size and the alphabet of streams such as tags and queries often grows fast, too. In this paper, we develop heuristic algorithms to tackle the space limitation. Specifically, we explore three approaches.

A frequent item based method (Section 3) has a precision of 100% and high recall, and uses $O(\frac{1}{\phi})$ space and $O(\log \frac{1}{\phi})$ time to process each item.

A hash-based method (Section 4) favors large θ, and has the space complexity $O(\frac{hb \log_b |\Sigma|}{\phi\theta})$ and per item time complexity $O(hb \log_b |\Sigma|)$, where b is the number of buckets of a hash function and h is the number of pairwisely independent hashes used in the algorithm.

Table 1 A running example.

S_1		y	w	y	x			x	x	z	z		x	z
S_2	x	w			w	y	w	y	y			z	y	w

x and z are discriminative items when $\theta = 3$ and $\phi = 0.1$

The hash-based method uses less space than the frequent item-based method when θ is large. The hash-based method also achieves a precision of 100% but the recall is worse than the frequent item-based method.

A hybrid method (Section 5) boosts the recall of the hash-based method, and consumes the least space among the three to achieve high precision and recall.

3 A frequent item based method

Since a discriminative item must be frequent in S_1 with respect to a threshold $\phi\theta n$, straightforwardly, we can employ any algorithms for finding frequent items on a single stream to first retrieve frequent items in S_1, and then remove false positives. Among numerous algorithms in the literature for finding frequent items, the space-saving algorithm [30] is the state-of-the-art method with low space complexity and high accuracy [10]. In this section, we first briefly review the space-saving algorithm, and then show how to extend it to find discriminative items.

3.1 The space-saving algorithm

Given a stream S whose current size is n, and a minimum support ϕn, the space-saving algorithm is a counter-based deterministic algorithm for finding items in S whose frequencies are no less than ϕn. The algorithm maintains a summary of the stream consisting of at most $m = \frac{1}{\phi}$ counters. The i-th counter $(e_i, c(e_i), \varepsilon(e_i))$ $(1 \leq i \leq m)$ records an item e_i being counted, the estimated count $c(e_i)$ of e_i, and the estimation error $\varepsilon(e_i)$. The m counters are sorted in the descending order of the estimated frequency c.

At the beginning, the counters are not associated with any item. When an item e is observed, if it is monitored in one of the m counters, the corresponding estimated count is incremented by 1. Otherwise, if there is a counter not associated with any item yet, then we assign the counter to e and initialize $c(e) = 1$ and $\varepsilon(e) = 0$. If all counters are associated with some items other than e, then e replaces e_m, which is the one with the least estimated frequency min, and sets $e_m = e$, $c(e_m) = min + 1$, and $\varepsilon(e_m) = min$.

Any item with a frequency exceeding ϕn must exist in the summary. Therefore, by reporting all items in the summary, the algorithm achieves 100% recall. For any item e_i $(1 \leq i \leq m)$ in the summary, its exact frequency $f(e_i)$ is bounded in the range $[c(e_i) - \varepsilon_i(e_i), c(e_i)]$. Thus, if $c(e_i) - \varepsilon(e_i) \geq \phi n$, e_i is guaranteed to have a frequency no less than the minimum support. By reporting the set of such guaranteed items, the algorithm achieves 100% precision.

Example 2 (*The space-saving algorithm* [30]) Assuming $\phi = 0.3$, let us find frequent items in S_1 in Table 1 with minimum frequency $10 \times \phi = 3$. We set up $\frac{1}{\phi} = 3$ counters.

After the first item y in the stream is read, counter $C_1 = (y, 1, 0)$ is set. After the first 6 items are read, i.e., *ywyxxx*, the content of the counters are $C_1 = (y, 2, 0)$, $C_2 = (w, 1, 0)$, and $C_3 = (x, 3, 0)$.

When we read the first z from S_1, C_2 is updated to $C_2 = (z, 2, 1)$. As the stream goes on, we sequentially update the counters as follows, $C_2 = (z, 3, 1)$, $C_3 = (x, 4, 0)$, and $C_2 = (z, 4, 1)$.

Finally, the content of the three counters are $C_1 = (y, 2, 0)$, $C_2 = (z, 4, 1)$, and $C_3 = (x, 4, 0)$. By checking the value of $c - \varepsilon$ in each counter against the minimum frequency support, x and z are reported as frequent items.

The space-saving algorithm requires space $O(\frac{1}{\phi})$. With a simple heap implementation of the stream summary, the algorithm processes every item in time $O(\log \frac{1}{\phi})$, and this can be improved to $O(1)$ by the Stream-Summary data structure [30].

3.2 Finding discriminative items

To find discriminative items in S_1 against S_2, we can run the space-saving algorithms on S_1 and S_2 separately and combine the information in the two summaries to discover discriminative items.

To be specific, we run the space-saving algorithm on S_1 to find items with frequency in S_1 no less than $\phi\theta n$. We also run the space-saving algorithm on S_2 to find items with frequency in S_2 no less than ϕn. Let E_i $(i = 1, 2)$ denote the set of items stored in the summary of the space-saving algorithm running on stream S_i.

If an item e is in the summary of S_i $(i = 1, 2)$, we denote the counter of e by $(e, c_i(e), \varepsilon_i(e))$. By the property of the space-saving algorithm, we have $c_i(e) - \varepsilon_i(e) \leq f_i(e) \leq c_i(e)$. Utilizing these upper and lower bounds of the frequencies of items in the summaries, we obtain the lower bound of the ratio.

Considering an item $e \in E_1$ such that $c_1(e) - \varepsilon_1(e) \geq \phi n$, e is guaranteed to be a discriminative item if it is in one of the following two cases.

Case 1 $e \notin E_2$. Because e is not in the summary of S_2, so $f_2(e) < \phi n$. We calculate the ratio $R(e) = \frac{f_1(e)}{f_2(e)} \geq \frac{\phi\theta n}{\phi n} = \theta$.

Case 2 $e \in E_2$ and $\frac{c_1(e) - \varepsilon_1(e)}{c_2(e)} \geq \theta$. Because $f_2(e) \leq c_2(e)$, so $R(e) = \frac{f_1(e)}{f_2(e)} \geq \frac{c_1(e) - \varepsilon_1(e)}{c_2(e)} \geq \theta$.

Clearly, by reporting the items in the above two cases, we achieve a precision of 100%. However, the recall of the above algorithm highly depends on the accuracy of the frequency bounds of the items. In general, in addition to the space-saving algorithm, any algorithm for finding frequent items can be used here as long as the algorithm can provide a bounded estimation of the frequencies of frequent items.

Example 3 (*The frequent item based method*) Consider the running example in Table 1. Let $\theta = 3$ and $\phi = 0.1$. We run the space-saving algorithm on S_1 to find items with minimum frequency $10\phi\theta = 3$. As shown in Example 2, x and z are frequent items in S_1 whose frequency lower bounds are 4 and 3, respectively. Similarly, we also find frequent items in S_2 with minimum frequency $10\phi = 1$. By checking x and z with respect to the two cases, we report that x and z are discriminative items.

3.3 Complexity analysis

Running the space-saving algorithms on S_1 and S_2 requires $O(\frac{1}{\phi\theta})$ and $O(\frac{1}{\phi})$ space, respectively. Hence, the frequent item based algorithm requires $O(\frac{1}{\phi\theta} + \frac{1}{\phi}) = O(\frac{1}{\phi})$ space. Importantly, the space complexity of the frequent item based method is independent from θ.

To update the summaries when a new item arrives, using a heap implementation, the algorithm spends $O(\log \frac{1}{\phi\theta} + \log \frac{1}{\phi}) = O(\log \frac{1}{\phi})$ time, while it can achieve $O(1)$ update time using the Stream-Summary data structure [30].

In many applications, we favor highly discriminative items and thus a large value of θ. Theorem 2 indicates that the number of discriminative items decreases as θ increases. There is potential to lower the space complexity when the value of θ is large. To take advantage of a large value of θ, we develop a hash-based method in the next section using space $O(\frac{\log|\Sigma|}{\phi\theta})$ which is better than the frequent item based method in space cost.

4 A hash-based method

In the frequent item based method, frequent items in S_1 and S_2 are computed independently. The frequent items in the two streams are compared only after the frequent item finding algorithm is completed on both streams. This late interaction of the two mining processes on the two streams may lead to counting many non-discriminative items. If an item x is frequent in S_1 and also very frequent in S_2, x will be counted in both streams. Can we try to let the two mining processes on the two streams communicate early so that the information that x is very frequent in S_2 can help to save the effort of counting x in S_1 and thus S_2? This is the motivation of the hash-based method.

4.1 Ideas

The following lemma helps us to identify a subset of items which may contain discriminative items.

Lemma 1 (Discriminative sets) *Let $T \subseteq \Sigma$ be a set of items. If*

$$\sum_{e\in T} f_1(e) \geq \theta \sum_{e\in T} f_2(e), \tag{1}$$

then T contains at least one item e such that $f_1(e) \geq \theta f_2(e)$.

Proof We prove by contradiction. Suppose for any item $e \in T$, $f_1(e) < \theta f_2(e)$. Then,

$$\sum_{e\in T} f_1(e) < \sum_{e\in T} \theta f_2(e) < \theta \sum_{e\in T} f_2(e),$$

resulting in a contradiction. □

For an item e, it may not be a discriminative item even if $f_1(e) \geq \theta f_2(e)$, since we constrain $f_1(e) \geq \theta\phi n$. However, Lemma 1 provides a necessary condition for finding discriminative items.

To utilize Lemma 1, once a set T of items is found to satisfy Formula (1), we recursively partition T into subsets until there is only one item e. Then, we check whether $f_1(e) \geq \theta\phi n$, if so, e is identified to be a discriminative item. We develop a hierarchical hashing structure to systematically manage the recursive partitioning.

4.2 Hierarchical hashing

Figure 1 illustrates the structure of the hierarchical hashing. A uniform hashing function with $b \ll |\Sigma|$ buckets on the alphabet Σ serves as the first level of the hierarchical hashing. A bucket B will be selected to expand to the next level if it satisfies our expanding criteria, which will be discussed in just a moment. Such a bucket is called a *discriminative bucket*.

For a discriminative bucket B, a different uniform hashing function is applied to the items hashed in B to construct the second level hashing. Then, those sub-buckets of B which are discriminative are recursively hashed into next level, forming the hierarchical hashing structure. We note that all hashing functions are uniform and each has b buckets. The number of distinct items hashed in every bucket is roughly equal. Thus, the hierarchical hashing has at most $\log_b |\Sigma|$ levels. On the $\log_b |\Sigma|$-th level, we directly put each item into a separate bucket, so that no conflicts can happen. The number of buckets on the last level may be slightly different from b.

A bucket B at a higher level is the *ancestor* of another bucket B' at a lower level if $B \supset B'$, that is, any item hashed into B' is hashed into B first. If B and B' are at two adjacent levels, B is also called the *parent* of B' and B' is a *child* of B. We call a bucket a *leaf* if it has no child. Please note that a leaf bucket may still have multiple items and may be expanded at a later time of the stream.

Each bucket B is associated with two counters $C_i(B)$ $(i = 1, 2)$ recording the occurrences of items from stream S_i hashed into the bucket from the time B is created until it is expanded into the next level child buckets. Therefore, the counters of a bucket are initialized to be 0 when the bucket is created, and are stopped being

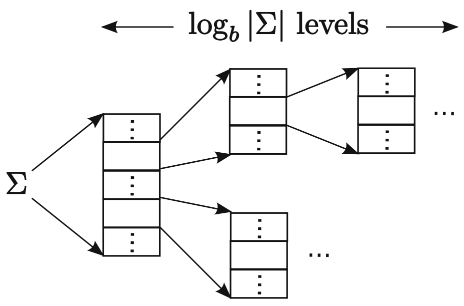


Figure 1 An illustration of the hierarchical hashing.

updated once the bucket is expanded. For a leaf bucket B, we can bound the sum of the frequencies of all items in B as

$$C_i(B) \leq \sum_{e \in B} f_i(e) \leq C_i(B) + \sum_{B' \in Anc(B)} C_i(B'),$$

where $Anc(B)$ is the set of all ancestor buckets of B.

To process a new item from stream S_i, the new item is hashed all the way down to the currently lowest level of the hierarchical hashing into the corresponding bucket B. The counter $C_i(B)$ is incremented. We note again that the counters of the ancestor buckets of B are not incremented. Only the bucket on the lowest level is updated.

Now, we present the expanding criteria that guides the hierarchical hashing to find discriminative items.

Lemma 2 (Discriminative buckets) *Given a bucket B, let $Anc(B)$ be the set of ancestor buckets of B. If*

$$C_1(B) \geq \theta(C_2(B) + \sum_{B' \in Anc(B)} C_2(B')), \tag{2}$$

then B contains at least one item e such that $f_1(e) \geq \theta f_2(e)$.

Proof For all items e hashed into B,

$$\sum_{e \in B} f_1(e) \geq C_1(B),$$

and, due to the construction of the hierarchical hashing,

$$\sum_{e \in B} f_2(e) \leq C_2(B) + \sum_{B' \in Anc(B)} C_2(B').$$

Then,

$$\sum_{e \in B} f_1(e) \geq \theta \sum_{e \in B} f_2(e).$$

By Lemma 1, this lemma follows immediately. □

Based on Lemma 2, we call a bucket B a *discriminative bucket* if B satisfies Formula (2) and $C_1(B) \geq \theta\phi n$. The condition $C_1(B) \geq \theta\phi n$ is to make sure that B is possible to contain frequent item in S_1.

A bucket which used to be a discriminative bucket may be disqualified from Lemma 2 as the streams continue. Given a bucket B, if none of its child buckets is discriminative at this moment, we delete all its child buckets and sum up their counters to B. In detail, let $Chi(B)$ denote the set of child buckets of B. The counter $C_i(B)$ $(i = 1, 2)$ of B is increased by $\sum_{B' \in Chi(B)} C_i(B')$. We note that the deleting procedure is always conducted bottom-up from the lowest level.

At the end, for an item e at the $\log_b |\Sigma|$-th level discriminative bucket, if $f_1(e) \geq \theta\phi n$, then, it is a discriminative item. By reporting all such items, the hash-based method has 100% precision.

A bucket that does not satisfy Formula (2) is still possible to contain discriminative items. To boost the recall, we adopt the common methedology of applying multiple independent hierarchical hashings to process the streams. The number of hierarchical hashing is determined empirically.

Example 4 (*The hash-based method*) Consider the running example in Table 1. Let $\theta = 3$, $\phi = 0.1$, and $b = 2$. At level one of the hierarchical hashing, assume x and y are hashed into bucket $B_{1,1}$, and z and w are hashed into bucket $B_{1,2}$. Table 2 shows the sequential updates of the counters of each bucket in the item arriving order.

After we read the first x from S_1, we detect that $B_{1,1}$ is discriminative. So it is expanded to buckets $B_{2,1}$ and $B_{2,2}$ at the second level, where x and y are hashed into $B_{2,1}$ and $B_{2,2}$, respectively.

Finally, we find x to be a discriminative item since $B_{2,1}$ satisfies Formula (2) and $C_1(B_{2,1})$ is larger than the minimum frequency support 3. However, the hash-based method does not report z as a discriminative item, because z is hashed into the same bucket with w and unfortunately w is very frequent in S_2. By a different hierarchical hashing, z might be hashed together with x, then, it can be found as a discriminative item.

In summary, the hash-based method consists of three steps, hashing items, growing hashing, and deleting buckets. Algorithm 1 presents the pseudo-code.

4.3 Complexity analysis

Since for a discriminative bucket B, $C_1(B) \geq \phi\theta n$, there are at most $\frac{1}{\phi\theta}$ discriminative buckets at each level of the hierarchical hashing. So the number of buckets in a single hashing structure is no more than $\frac{b \log_b |\Sigma|}{\phi\theta}$. Assume h hierarchical hashing structures are used concurrently. The hash-based method uses $O(\frac{hb \log_b |\Sigma|}{\phi\theta})$ space in the worst case. In practice, the space is much smaller since the number of discriminative buckets is much less than $\frac{1}{\phi\theta}$.

To process an item in a single hierarchical hashing structure, the hashing procedure takes time $O(\log_b |\Sigma|)$. The deleting procedure spends at most $O(b \log_b |\Sigma|)$ time from bottom up. Therefore, the update time of the hash-based method is at most $O(hb \log_b |\Sigma|)$. The hash-based method also runs much faster in practice because the deleting procedure does not happen often.

Table 2 A running example of the hash-based method.

$B_{1,1}$	C_1	0	1	1	2										
$\{x, y\}$	C_2	1	1	1	1										
$B_{2,1}$	C_1					1	1	1	2	3	3	3	3	4	4
$\{x\}$	C_2					0	0	0	0	0	0	0	0	0	0
$B_{2,2}$	C_1					0	0	0	0	0	0	0	0	0	0
$\{y\}$	C_2					0	1	1	2	3	3	3	3	4	4
$B_{1,2}$	C_1	0	0	1	1	1	1	1	1	1	2	3	3	3	4
$\{z, w\}$	C_2	0	1	1	1	2	2	3	3	3	3	3	4	4	5

Algorithm 1 The hash-based method.

```
Input:  two streams $S_1$ and $S_2$; parameters $\phi$ and $\theta$;
Output:  the set $E$ of discriminative items;
Description:
 1: construct $h$ independent hierarchical hashing structures and initialize their first
    level buckets;
 2: for all item $e \in S_i$ $(i = 1, 2)$ do
 3:   for all hierarchical hashing $H$ do
 4:     /* hashing items */
        let $B$ be the bucket on the first level of $H$ where $e$ is hashed into;
 5:     while $B$ is not a leaf bucket do
 6:       assume $e$ is hashed into the child bucket $B'$ of $B$;
 7:       $B = B'$;
 8:     end while
 9:     $C_i(B) = C_i(B) + 1$;
        /* growing hashing */
10:     if $B$ is discriminative (Lemma 2) and $B$ is not on the $\log_b |\Sigma|$-th level then
11:       apply a uniform hash with $b$ buckets on the items of $B$ to construct the
          next level hashing;
12:     end if
        /* deleting buckets */
13:     if $e$ is from stream $S_2$ then
14:       let $B_p$ be the parent of $B$;
15:       while non of $B_p$'s child bucket is discriminative do
16:         $C_i(B_p) = \sum_{B_c \in Chi(B_p)} C_i(B_c)$;
17:         delete all child buckets of $B_p$;
18:         assign $B_p$ the parent of $B_p$;
19:       end while
20:     end if
21:   end for
22: end for
23: return all items in the discriminative buckets on the $\log_b |\Sigma|$ level;
```

5 A hybrid method

One drawback of the hash-based method is that a number of discriminative items may be buried in non-discriminative buckets, so that the recall of a single hierarchical hashing structure is low. In this section, we aim at discovering those concealed discriminative items to improve the recall of a single hierarchical hashing structure. By doing this, we are able to achieve the same recall while reducing the number of hierarchical hashing structures, compared to the hash-based method.

5.1 The method

To discover the concealed discriminative items in a non-discriminative bucket B (i.e., a bucket that does not satisfy Formula (2)), we need a more aggressive expanding criteria to grow the hierarchical hashing to deal with non-discriminative buckets.

Given a bucket, the items hashed into this bucket can be viewed as the sub-streams of streams S_1 and S_2, respectively. We build a space-saving summary on the sub-stream of S_1 flowing through B. Thus, by this hybrid structure, we can capture items in B which are frequent in S_1. Then, by expanding B, we have a good chance to discover discriminative items hidden in the non-discriminative bucket. Figure 2 illustrates the idea.

To be concrete, for each leaf bucket B, the hybrid method maintains a space-saving summary with $k \ll \frac{|\Sigma|}{b}$ counters for items from S_1 hashed into B. When B is selected to be expanded to the next level in the hashing growing phase, the counters of an item $e \in B$ are forwarded to the corresponding child bucket of B where e is hashed into. Therefore, the space-saving summaries are only kept in leaf buckets. Any intermediate bucket does not keep such a summary. For an item e kept in the summary of S_1, in addition to its counter $(e, c_1(e), \varepsilon_1(e))$ of S_1, we maintain another counter $c_2(e)$ to record the number of occurrences of e in S_2 once it is recorded by the summary of S_1. By doing this, inequality $c_1(e) - \varepsilon_1(e) \leq f_1(e) \leq c_1(e)$ still holds.

Using the space-saving summary, we handle discriminative buckets in a slightly different way from the process in the hash-based method. For a bucket B, according to the space-saving algorithm introduced in Section 3.1, the k counters are initially filled with the first k distinct items coming into B. In the hash-based method, a bucket B is expanded immediately once it is found to be discriminative. However, the k counters of B already record the top-k most frequent items in B. It is not necessary to expand B if the k counters are not all occupied. Therefore, we delay expanding a discriminative bucket B until all k counters are used. In the process of expanding a discriminative bucket, the existing counters are simply forwarded to its child buckets.

To handle non-discriminative buckets, in the cases where all k counters of a bucket B are used and B has not been found discriminative at the moment, we adopt a more aggressive expanding criteria. When a new item comes in B and it is different from the k items in the summary, let e be the item with the minimum estimated frequency in S_1 among the k items. We expand B to the next level if $c_1(e) - \varepsilon_1(e) \geq \max\{\phi\theta n, \mu\}$. Here, μ is a controlling parameter which is set to $\sqrt{\theta}$ empirically. The rationale behind this expanding criteria is that the k items kept in the summary have high possibility to be discriminative items, as they are frequent in S_1.

Finally, to report discriminative items, we check every leaf bucket B in the hierarchical hashing structure. For each counter $(e, c_1(e), \varepsilon(e))$ kept in B, we report e as a discriminative item if $\frac{c_1(e)-\varepsilon(e)}{c_2(e)} \geq \theta$. Although $f_1(e) \geq c_1(e) - \varepsilon(e)$, we cannot guarantee that $f_2(e) \leq c_2(e)$, thus $R(e) \geq \theta$ is not assured. However, the recall is

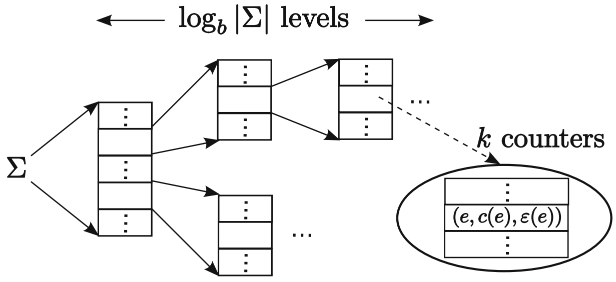


Figure 2 An illustration of the hybrid method.

improved. Essentially, the hybrid method trades precision for recall. Our experiments in Section 6 verify that this trade-off is beneficial.

In the same way as the hash-based method, multiple hierarchical hashing structures can be applied.

Example 5 (*The hybrid method*) For the running example in Table 1, the hash-based method shown in Example 4 cannot find z as a discriminate item, since z is concealed by the effect of w. To tackle this problem, the hybrid method runs a space saving algorithm on S_1 with 1 counter. Then, it will find that z is frequent in S_1 and expand bucket $B_{1,2}$. At the end, the space-saving counter of z is $(z, 4, 1)$ and the additional counter of z on stream S_2 is $c_2(z) = 1$. Thus, z is found to be discriminative.

5.2 Complexity analysis

In a single hierarchical hashing, to store the space-saving summaries, the hybrid method requires at most $O(\frac{bk}{\phi\theta})$ space more than the hash-based method, since there are no more than $O(\frac{b}{\phi\theta})$ leaf buckets. So the space complexity of the hybrid method is

$$O\left(\frac{hb\,\log_b|\Sigma|}{\phi\theta}\right) + O\left(\frac{hb\,k}{\phi\theta}\right) = O\left(\frac{hb\,(\log_b|\Sigma| + k)}{\phi\theta}\right),$$

which is the same as the hash-based method. However, to achieve the same recall, the hybrid method reduces the number of hierarchical hashing needed. Therefore, the hybrid method is expected to outperform the hash-based method in terms of space usage.

The hybrid method needs to update both the hierarchical hashing and the space-saving summaries. With a heap implementation for the space-saving summaries, its time complexity is $O(h(b\,\log_b|\Sigma| + \log k))$, and $O(hb\,\log_b|\Sigma|)$ with the Stream-Summary data structure [30].

6 Empirical studies

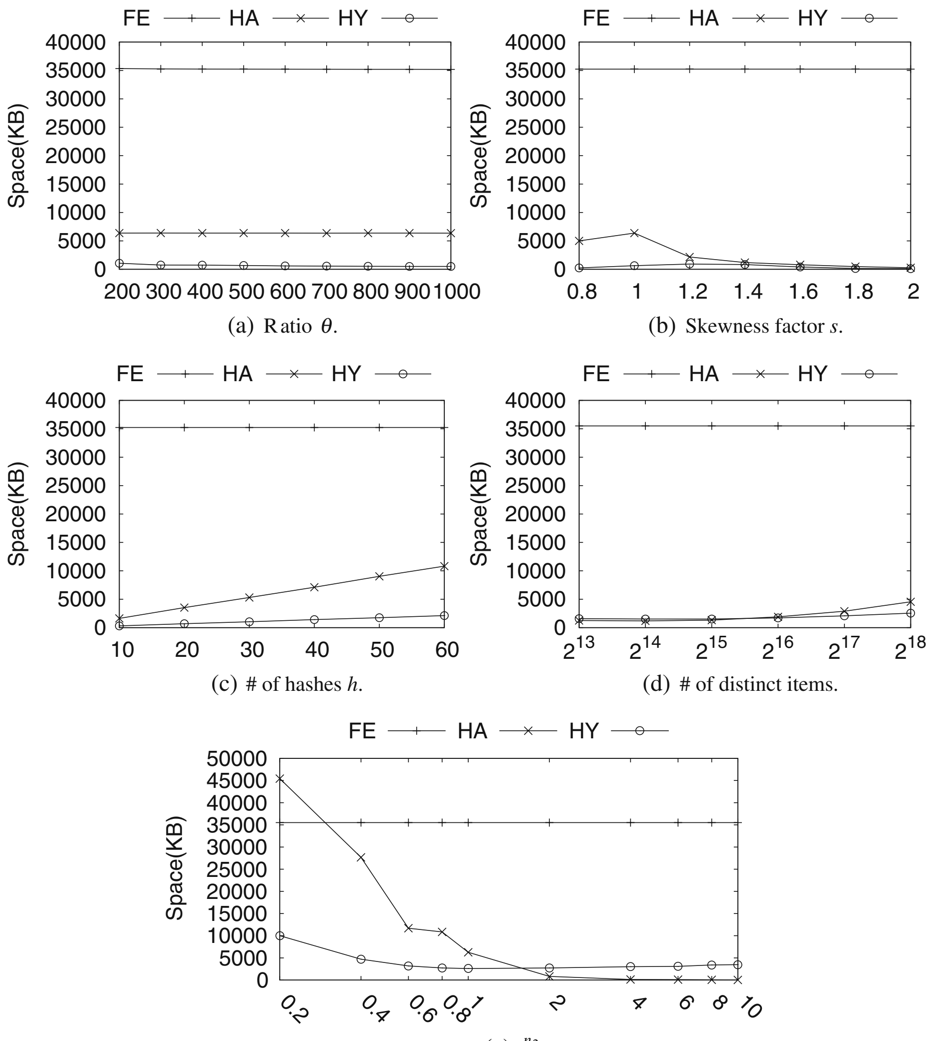
We conducted experiments on real and synthetic data sets to evaluate the accuracy and efficiency of our three methods,[1] the frequent item based method (FE), the hash-based method (HA), and the hybrid method (HY). The space-saving algorithm used in FE and HY was implemented using heap rather than the Stream-Summary structure. The performance of our algorithms is not sensitivc to the minimum support threshold ϕ. So we set $\phi = 10^{-6}$ and does not change it in the experiments. For HA and HY, the hash fanout b is set to 32 all the time, and the number of counters used in each bucket in HY is $k = 5$. The hash functions we use are pairwisely independent and implemented by the method stated in [8].

All methods were implemented in C++ and compiled by Microsoft Visual Studio 2008. Experiments were conducted on a desktop computer with an Intel Core 2 Duo E8400 3GHz CPU and 4GB main memory running 64bit Microsoft Windows XP.

[1]The source code of the three methods can be downloaded at http://www.cs.sfu.ca/∼bjiang/personal/discriminative_item_code.zip.

6.1 Synthetic data

We generated two streams in Zipfian distribution with skewness factor s varying from 0.8 to 2. The size of each stream is 1,000,000 drawn from the alphabet Σ whose size is $2^{20} \approx 1{,}000{,}000$. We also ensure that there are a set of frequent items in S_1 also being frequent in S_2, so that the set of discriminative items is not trivially equivalent to the set of frequent items in S_1. To do this, we select items with frequencies over 100 from S_1 and randomly choose 25% of them so that their frequencies in S_2 also exceeding 100. By default, the ratio parameter $\theta = 500$, the skewness factor $s = 1$, and the number of hashes are 35 and 18 for HA and HY,



(a) Ratio θ.

(b) Skewness factor s.

(c) # of hashes h.

(d) # of distinct items.

(e) $\frac{n_2}{n_1}$.

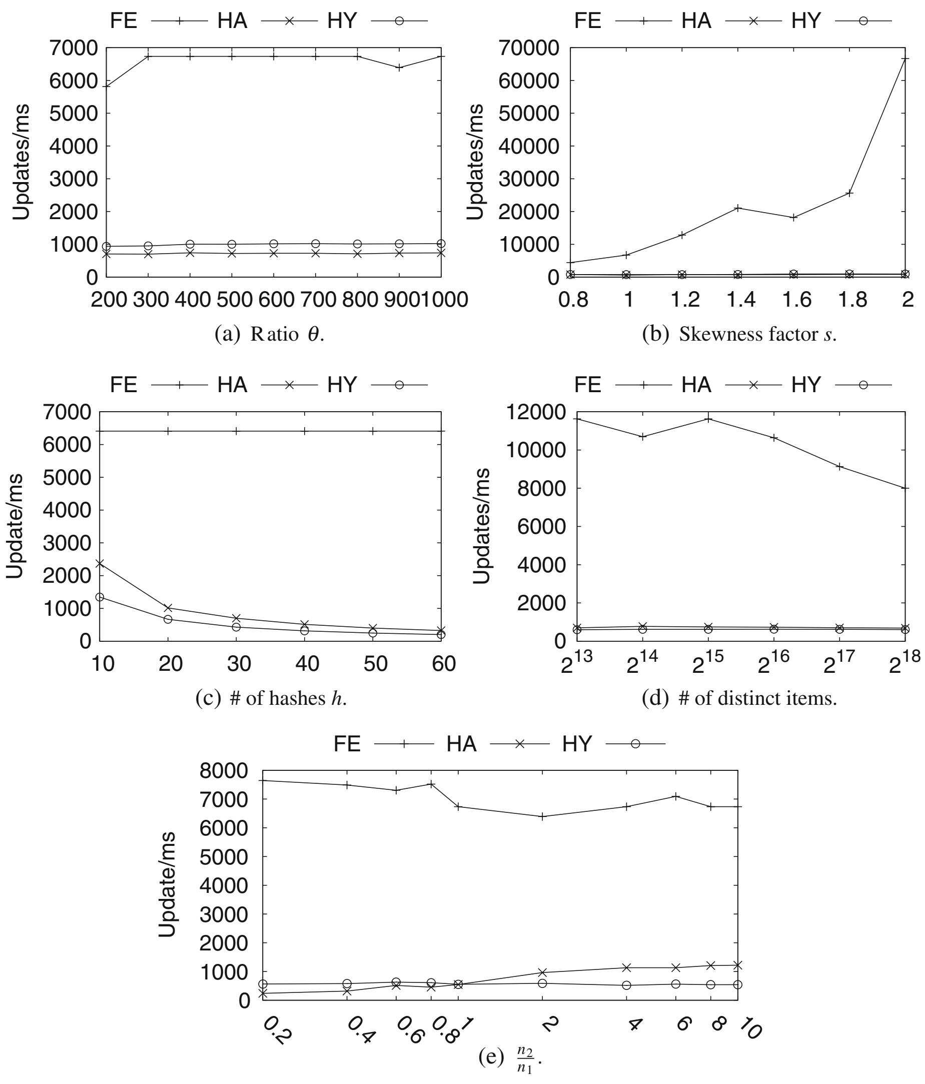
Figure 3 Space on synthetic data sets.

respectively, which are selected from our experiment results to balance accuracy and efficiency.

We conduct experiments to test the efficiency and accuracy of our three methods with respect to the ratio parameter θ, the skewness factor s, the number of hashes h, the number of distinct items in the two streams, and the value of $\frac{n_2}{n_1}$.

6.1.1 Efficiency

Figure 3 compares the space usage in the three methods. FE uses the most space among the three. It is only dependent on the minimum support threshold ϕ and

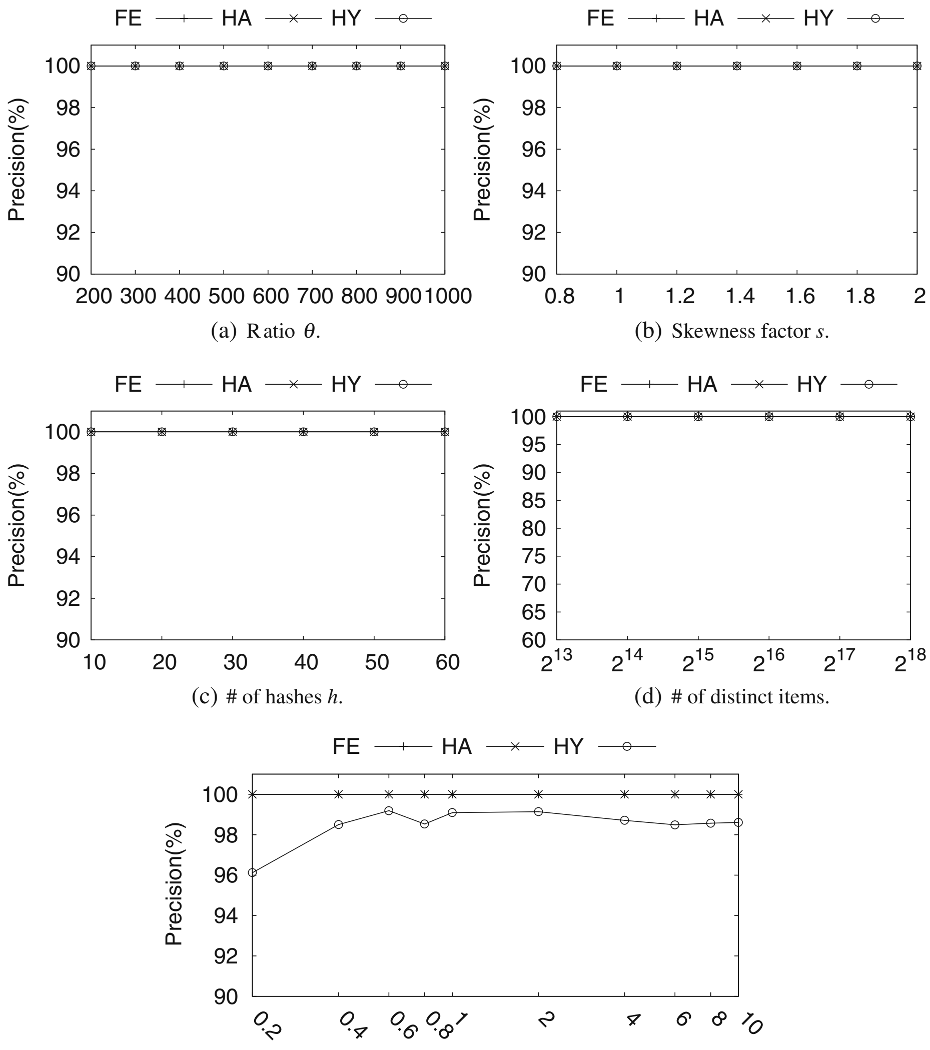


(a) Ratio θ. (b) Skewness factor s. (c) # of hashes h. (d) # of distinct items. (e) $\frac{n_2}{n_1}$.

Figure 4 Number of updates per ms on synthetic data sets.

invariant to the ratio parameter θ, the skewness factor s, the number of distinct items, and $\frac{n_2}{n_1}$.

HA uses only about $\frac{1}{5}$ space of FE. Although the space complexity of HA is $O(\frac{hb \log_b |\Sigma|}{\phi\theta})$, Figure 3a shows that its space usage is not sensible to θ, because the number of discriminative buckets is far less than $\frac{1}{\phi\theta}$. We also see that the space usage of HA is small on data sets of large skewness factors, where the number of discriminative items is small. The space usage of HA increases linearly with respect to the number of hierarchical hashing. It also increases with respect to the number of

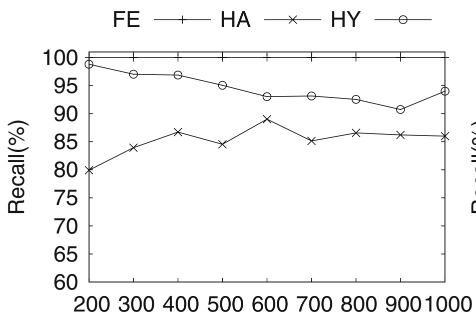


(a) Ratio θ. (b) Skewness factor s. (c) # of hashes h. (d) # of distinct items. (e) $\frac{n_2}{n_1}$.

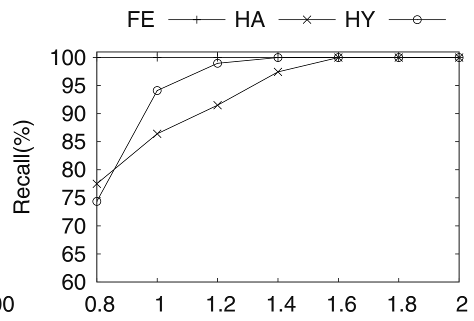
Figure 5 Precision on synthetic data sets.

distinct items, since more distinct items would expand more buckets. But it decreases when $\frac{n_2}{n_1}$ increases, since the number of expanded buckets decreases.

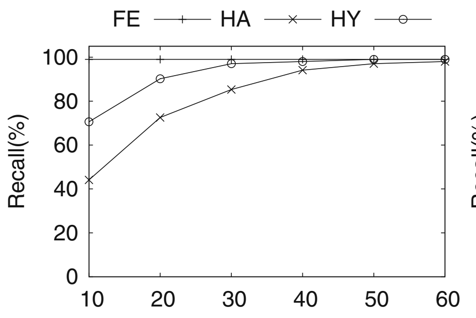
HY is the most space-efficient method which outperforms FE by tens of times. HY also beats HA by several times. Figure 3c shows that even using the same number of hierarchical hashing, HY uses less space than HA. Because expanding a discriminative bucket in HY is delayed until all k counters are filled, HY may have less expanded discriminative buckets than HA thus reduces space usage. The space usage of HY is not very sensitive to θ and s, while it also has a linear increasing trend with respect to the number of hierarchical hashing. HY has similar trends as HA



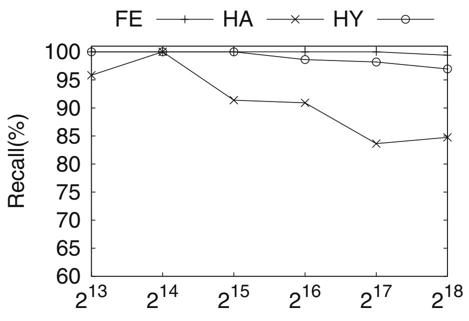
(a) Ratio θ.



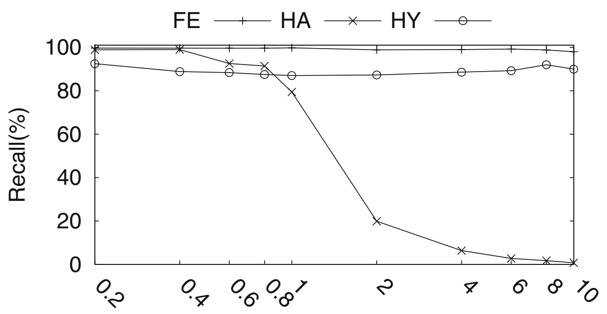
(b) Skewness factor s.



(c) # of hashes h.



(d) # of distinct items.



(e) $\frac{n_2}{n_1}$.

Figure 6 Recall on synthetic data sets.

Table 3 Topics in real data sets.

Data set	Partition P_1	Partition P_2
Wikipedia	Mathematics	Law
Newsgroups	comp.graphics	alt.atheism
	comp.sys.ibm.pc.hardware	rec.autos
	comp.sys.mac.hardware	rec.motorcycles
	comp.os.ms-windows.misc	rec.sport.baseball
	comp.windows.x	rec.sport.hockey
	misc.forsale	soc.religion.christian
	sci.crypt	talk.politics.guns
	sci.electronics	talk.politics.mideast
	sci.med	talk.politics.misc
	sci.space	talk.religion.misc

with respect to the number of distinct items and $\frac{n_2}{n_1}$, since they share the hierarchical hashing structure.

The runtime is plotted in Figure 4. FE is the fastest method which can process more than 6,000 items per millisecond. It can even handle more than 60,000 items on data sets with skewness factor $s = 2$. In Figure 4b, we see that FE runs faster in more skewed data sets. This is due to the heap implementation of the space-saving algorithm. We can expect a stable performance with the Stream-Summary implementation. FE also runs faster when the number of distinct items is small, since in this case the summary does not change often.

HA and HY can support around 1,000 updates per millisecond. Figure 4a shows that HY is slightly faster than HA, since HY uses less hierarchical hashing than HA. When using the same number of hashing, Figure 4c shows that HY is slower than HA, as HY needs to maintain the space-saving summary.

6.1.2 Accuracy

Figure 5 compares the precision of the three methods. As stated in Sections 3 and 4, FE and HA are guaranteed to have 100% precision. We see that the precision of *HY* is also close to 100%, and there is no clear trend related to the number of distinct items and $\frac{n_2}{n_1}$.

In terms of recall, Figure 6 shows that FE has a recall of almost 100%. The recalls of HA and HY also increase to 100% as the skewness factor increases or using more hierarchical hashing. HY has a better recall than HA in most cases, even when HY uses only a half number of hierarchical hashing. In Figure 6a, the recall of HA increases slowly as θ increases, however, the recall of HY decreases. When θ is large, there are less expanded buckets since the expanding criteria of non-discriminative buckets is controlled by the parameter $\mu = \sqrt{\theta}$.

Figure 6e shows that the recall of HA decreases dramatically as $\frac{n_2}{n_1}$ increases over 1, since the number of expanded buckets decreases a lot because items from S_2 flood

Table 4 Size of the real data sets in words.

Wikipedia		Newsgroups	
P_1	P_2	P_1	P_2
3,676,073	3,851,345	2,637,816	2,914,446

Table 5 Top-5 most discriminative words in the Wikipedia data set.

Mathematics against law		Law against mathematics	
Words	Ratio	Words	Ratio
Polynomial	855.55	Constitution	1,010.25
Algebra	761.14	Jurisdiction	894.75
Algebraic	703.65	Justice	480.38
Geometry	679.17	Parliament	448.58
Topology	645.50	Defendant	442.86

the buckets and make them difficult to expand. However, as a remedy, when $\frac{n_2}{n_1}$ is larger than 1, we could duplicate every item from S_1 $\frac{n_2}{n_1}$ times it is observed it so that S_1 has similar size as S_2. Then, we also scale θ to $\frac{n_2}{n_1}\theta$ correspondingly. So, the set of discriminative does not change while HA can work well on the duplicated streams.

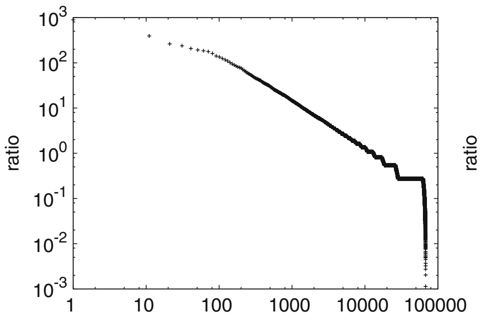
6.2 Real data

We use two real data sets, namely the Wikipedia data set and the 20 Newsgroups data set, obtained from http://en.wikipedia.org/ and http://people.csail.mit.edu/jrennie/20Newsgroups/, respectively. In the Wikipedia data set, we obtain 5,000 articles on the topic of mathematics and 4,000 articles on the topic of law. Articles on the same topic are merged into one stream.

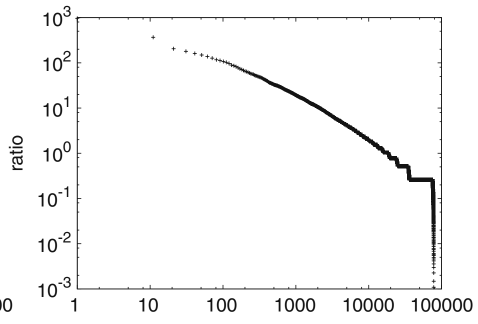
The 20 Newsgroups data set consists of 18, 846 newsgroup documents, partitioned evenly across 20 different newsgroups, each corresponding to a different topic shown in Table 3. We divide the 20 newsgroups into to 2 partitions as shown in Table 3, such that the topics in one partition are closely related to each other. Articles in the same partition then are merged into a single stream.

For all articles, we only conduct stemming but do not filter out stopping words. Table 4 lists the size of the two partitions of each data set.

Table 5 lists the top-5 high ratio words in the Wikipedia data set, which match our common intuition. Figures 7 and 8 show the ratio distribution of all terms on the two real data sets in log-log graph. We observe a power law distribution of the ratio. The sharp tails are caused by the minimum support threshold ϕ.



(a) $S_1 = P_1$ and $S_2 = P_2$.



(b) $S_1 = P_2$ and $S_2 = P_1$.

Figure 7 The distribution of ratio on the Wikipedia data set.

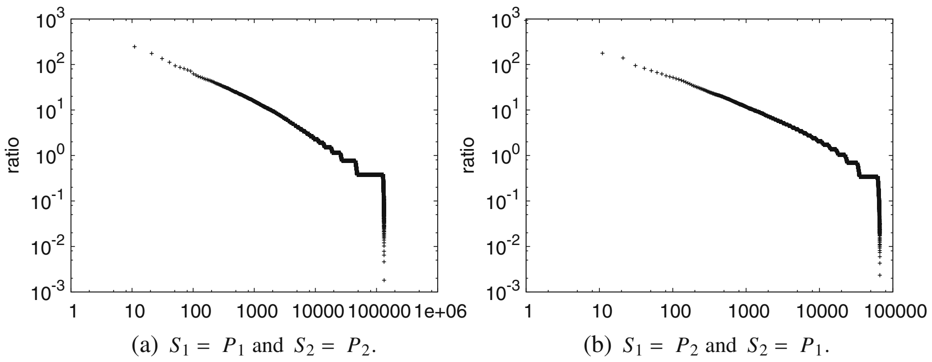


Figure 8 The distribution of ratio on the Newsgroups data set.

Tables 6 and 7 show the space and time usage, the precision, and the recall of the three methods on the two real data sets. We fix the number of hierarchical hashing used in HA and HY to 30 and 2, respectively. The trends are consistent with those on the synthetic data sets.

FE always has a precision and a recall of 100% on the real data sets, and the fastest update time per item. However, the space usage in FE is 2 orders of magnitude larger than the other two methods. HA has a low recall on the real data sets. HY can achieve the comparable precision and recall to FE, at the same time, use much smaller space.

7 Related work

The problem of finding discriminative items between two streams can provide solutions to many Web mining applications such as tag suggestions [16, 21, 28, 34], web document summarization [23, 26, 38], email summarization [7, 27], web search result summarization [24], search engine query analysis [36], and social network

Table 6 Results on the Wikipedia data set.

θ		Space (MB)	Updates/ms	Precision (%)	Recall (%)
$S_1 = P_1$ and $S_2 = P_2$					
100	FE	34.68	20,967.74	100	100
	HA	0.97	1,068.17	100	56.64
	HY	0.29	5,803.71	100	89.51
300	FE	34.44	20,967.74	100	100
	HA	0.96	1,063.50	100	61.11
	HY	0.22	5,803.71	100	83.33
$S_1 = P_2$ and $S_2 = P_1$					
100	FE	34.68	20,073.11	100	100
	HA	1.72	1,082.61	100	59.29
	HY	0.50	5,668.24	97.87	81.42
300	FE	34.44	19,301.07	100	100
	HA	1.71	1,087.46	100	69.23
	HY	0.51	5,732.99	100	84.62

Table 7 Results on the Newsgroups data set.

θ		Space (MB)	Updates/ms	Precision (%)	Recall (%)
$S_1 = P_1$ and $S_2 = P_2$					
100	FE	34.68	17,795.71	100	100
	HA	3.24	981.49	100	59.57
	HY	0.65	5,470.21	91.67	93.62
200	FE	34.50	16,927.63	100	100
	HA	3.23	973.57	100	68.75
	HY	0.36	5,552.26	93.75	93.75
$S_1 = P_2$ and $S_2 = P_1$					
100	FE	34.68	16,876.18	100	100
	HA	1.12	984.27	100	33.33
	HY	0.48	5,228.12	82.35	93.33
200	FE	34.50	16,140.30	100	100
	HA	1.11	989.71	100	42.86
	HY	0.30	5,464.82	77.78	100

analysis [32, 35]. An essential issue inherent in those applications is to find discriminative tags or keywords that can distinguish the target object from many others. Statistically, we model such discrimination in frequency ratio. Due to the large amount of data arriving or being generated in high speed on the Web, the streaming model is appropriate.

Discriminative items are highly related to frequent items in data streams and emerging patterns in pattern mining. We review the studies on mining frequent items in data streams and mining emerging patterns in Section 7.1 and 7.2, respectively.

7.1 Finding frequent items in data streams

The problem of finding frequent items is extensively studied in data stream community since 1980s due to its intuitive interest and importance. In the literature, there are various formulations of this problem, including finding top-k most frequent items [9, 30, 31], finding all frequent items with respect to a user-specified frequency threshold [2, 22, 25], finding frequent items over sliding windows [2, 13, 25, 31], and so on. Among all these formulations, the problem of finding all frequent items with respect to a user-specified frequency threshold is the one most relevant to our problem. We review the major algorithms of this problem in detail.

Formally, given a stream S of length n and a threshold ϕ, the goal is to return a set of items E so that for each $e \in E$, the frequency $f(e) \geq \phi n$. Unfortunately, any online algorithm that finds the exact set E must use $\Omega(|\Sigma| \log \frac{n}{|\Sigma|})$ space in the worst case [22], where Σ denotes the alphabet. To overcome this lower bound, the problem of finding ϵ-approximate frequent items [29, 30] is introduced. The goal is to find a set of items $\hat{E}$ where each item $e \in \hat{E}$ satisfies $f(e) > (\phi - \epsilon)n$.

Cormode and Hadjieleftheriou [10] compared several algorithms on this subject, and divided them into three classes, namely, counter-based algorithms [6, 14, 22, 29, 30], quantile algorithms [20, 29], and randomized sketch algorithms [1, 11, 12]. Besides finding frequent items, counter-based algorithms can also estimate their frequencies. We note that any counter-based algorithm can be plugged into the framework of our frequent item-based method for finding discriminative items.

Below, we summarize several representative counter-based algorithms, which are the frequent algorithm, the lossy counting algorithm, and the space-saving algorithm.

The frequent algorithm was discovered independently by [22] and [14], which generalized the majority algorithm [6, 18] of finding the item whose frequency exceeds $n/2$. The central idea of these algorithms is "cancelation". The algorithms maintain $\frac{1}{\phi} - 1$ (item, count) pairs, whose counts are initialized to 0. If a new observation e is an item in these pairs, increment the corresponding count by 1. Else, if there is some pair with 0 count, then allocate this pair to this item and set it to $(e, 1)$; otherwise, decrement the counts of all pairs by 1. It can be proved that any item with frequency exceeding ϕn must be kept in these pairs when the algorithms terminate. Besides output of a super set of frequent items, the count associated with each item is at most ϕn below the true frequency. However, in practice, it's not suitable to be used for frequency estimation [10]. Space used by the frequent algorithms is $O(1/\phi)$, which is independent from the size of the stream.

The lossy counting algorithm proposed by [29] stores tuples (j, l_j, δ_j) where j is an element from Σ, l_j is the lower bound of j's count, and δ_j satisfies $l_j \leq f(j) \leq l_j + \delta_j$. When an item j arrives, if j is stored, then increment its lower bound by 1. Otherwise, create a new tuple $(j, 1, \lfloor \phi n \rfloor)$. From time to time, the algorithm deletes tuples with $l_j + \delta_j < \phi n$. Like the frequent algorithm, when the algorithm terminates, all items with frequencies larger than ϕn are stored and the error of frequency estimation is within ϕn for any item. There is a nice property that highly frequent items, if they appear early in the stream, have very accurate estimated frequencies. In terms of space usage, the algorithm requires $O(\frac{1}{\phi} \log \phi n)$ in the worst case.

The space-saving algorithm [30] described in Section 3.1 is also a one-pass counter-based algorithm similar to the lossy counting algorithm and shares the same nice property that the items stored by the algorithm early in the stream and not removed later have very accurate estimated frequencies. Experiments in [10] indicate that the space-saving algorithm outperforms other algorithms in terms of precision, recall, and space usage.

7.2 Emerging patterns

We note that emerging patterns (EP for short) studied in [3, 15, 17, 37] is another notion that discovers the discriminative items/patterns in one data set against the other. Given two transaction data sets, emerging patterns are those itemsets whose supports in one data set are significantly larger than their supports in the other. Formally, let $supp_i(X)$ denote the support of X in dataset i $(i = 1, 2)$ and define *growth rate* $GR(X)$ of an item set X as

$$GR(X) = \begin{cases} 0, & \text{if } supp_1(X) = 0 \text{ and } supp_2(X) = 0 \\ \infty, & \text{if } supp_1(X) = 0 \text{ and } supp_2(X) \neq 0 \\ \frac{supp_1(X)}{supp_2(X)}, & \text{otherwise} \end{cases}$$

Given $\phi > 1$, the goal is to find emerging patterns with growth rates more than ϕ. These emerging patterns are called ϕ-emerging patterns.

Emerging patterns as itemsets inherently have a combinatoric nature. Algorithms for finding emerging patterns spend most effort on addressing the problem of compactly representing emerging patterns and efficient manipulation of emerging patterns to avoid enumerating all possible itemsets. Mining discriminative items does not need to deal with this issue. Furthermore, the problem of emerging patterns are considered in a static transaction database environment. We position our problem in the data stream settings and focus on reducing memory space of use. To the best of our knowledge, we are the first to study the problem of finding discriminative items between two data streams.

8 Conclusions

In this paper, motivated by a class of Web mining applications including tagging Web objects, summarizing web documents, and analyzing search queries, we tackle the problem of finding discriminative items between streams, which are frequent in one stream but infrequent in another. We prove a space lower bound of exactly finding all discriminative items and develop three heuristic algorithms that by one scan can achieve high precision and recall using sub-linear space and sub-linear processing time per item with respect to the size of the alphabet. The complexity of all algorithms are independent from the size of streams.

References

1. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. In: Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC'96), pp. 20–29 (1996)
2. Arasu, A., Manku, G.S.: Approximate counts and quantiles over sliding windows. In: Proceedings of 23th ACM SIGMOD Principles of Database Systems (PODS'04), pp. 286–296 (2004)
3. Bailey, J., Manoukian, T., Ramamohanarao, K.: Fast algorithms for mining emerging patterns. In: Proceedings of 6th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'02), pp. 39–50 (2002)
4. Blei, D.M., Lafferty, J.D.: Dynamic topic models. In: Proceedings of the 23rd International Conference (ICML'06), ACM, ACM International Conference Proceeding Series, vol. 148, pp. 113–120 (2006)
5. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. J. Mach. Learn. Res. **3**, 993–1022 (2003)
6. Boyer, R.S., Moore, J.S.: Mjrty: a fast majority vote algorithm. In: Automated Reasoning: Essays in Honor of Woody Bledsoe, pp. 105–118 (1991)
7. Carenini, G., Ng, R.T., Zhou, X.: Summarizing email conversations with clue words. In: Proceedings of 16th International World Wide Web Conference (WWW'07), pp. 91–100 (2007)
8. Carter, L., Wegman, M.N.: Universal classes of hash functions (extended abstract). In: Proceedings of the 9th Annual ACM Symposium on Theory of Computing (STOC'77), pp. 106–112 (1977)
9. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. In: Proceedings of 29th International Colloquium on Automata, Languages and Programming (ICALP'02), pp. 693–703 (2002)
10. Cormode, G., Hadjieleftheriou, M.: Finding frequent items in data streams. In: Proceedings of the VLDB Endowment (PVLDB'08), vol. 1, no. 2, pp. 1530–1541 (2008)
11. Cormode, G., Muthukrishnan, S.: What's new: finding significant differences in network data streams. In: Proceedings of 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'04) (2004)
12. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. J. Algorithms **55**(1), 58–75 (2005)

13. Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows (extended abstract). In: Proceedings of 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'02), pp. 635–644 (2002)
14. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: Proceedings of 10th European Symposium on Algorithms (ESA'02), pp. 348–360 (2002)
15. Dong, G., Li, J.: Efficient mining of emerging patterns: discovering trends and differences. In: Proceedings of 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'99), pp. 43–52 (1999)
16. Dubinko, M., Kumar, R., Magnani, J., Novak, J., Raghavan, P., Tomkins, A.: Visualizing tags over time. In: Proceedings of 15th International World Wide Web Conference (WWW'06), pp. 193–202 (2006)
17. Fan, H., Ramamohanarao, K.: An efficient single-scan algorithm for mining essential jumping emerging patterns for classification. In: Proceedings of 6th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'02), pp. 456–462 (2002)
18. Fischer, M.J., Salzberg, S.L.: Finding a majority among n votes. J. Algorithms **3**, 376–379 (1982)
19. Ganguly, S.: Lower bounds on frequency estimation of data streams. In: Proceedings of 3rd International Computer Science Symposium in Russia (CSR'08). Lecture Notes in Computer Science, vol. 5010, pp. 204–215. Springer (2008)
20. Greenwald, M., Khanna, S.: Space-efficient online computation of quantile summaries. In: Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD'01), pp. 58–66 (2001)
21. Halpin, H., Robu, V., Shepherd, H.: The complex dynamics of collaborative tagging. In: Proceedings of 16th International World Wide Web Conference (WWW'07), pp. 211–220 (2007)
22. Karp, R.M., Shenker, S., Papadimitriou, C.H.: A simple algorithm for finding frequent elements in streams and bags. ACM Trans. Database Syst. **28**, 51–55 (2003)
23. Khy, S., Ishikawa, Y., Kitagawa, H.: A novelty-based clustering method for on-line documents. World Wide Web **11**(1), 1–37 (2008)
24. Kuo, B.Y.L., Hentrich, T., Good, B.M., Wilkinson, M.D.: Tag clouds for summarizing web search results. In: Proceedings of 16th International World Wide Web Conference (WWW'07), pp. 1203–1204 (2007)
25. Lee, L.K., Ting, H.F.: A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In: Proceedings of 25th ACM SIGMOD Principles of Database Systems (PODS'06), pp. 290–297 (2006)
26. Li, L., Zhou, K., Xue, G.R., Zha, H., Yu, Y.: Enhancing diversity, coverage and balance for summarization through structure learning. In: Proceedings of 18th International World Wide Web Conference (WWW'09), pp. 71–80 (2009)
27. Li, W., Zhong, N., Yao, Y., Liu, J.: An operable email based intelligent personal assistant. World Wide Web **12**(2), 125–147 (2009)
28. Liu, D., Hua, X.S., Yang, L., Wang, M., Zhang, H.J.: Tag ranking. In: Proceedings of 18th International World Wide Web Conference (WWW'09), pp. 351–360 (2009)
29. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In: Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02), pp. 346–357. Morgan Kaufmann (2002)
30. Metwally, A., Agrawal, D., Abbadi, A.E.: Efficient computation of frequent and top-k elements in data streams. In: Proceedings of 10th International Conference on Database Theory (ICDT'05), pp. 398–412 (2005)
31. Pingda, S., Huahui, C.: A new method to find top k items in data streams at arbitrary time granularities. In: Proceedings of 2008 International Conference on Computer Science and Software Engineering (CSSE'08), vol. 4, pp. 267–270 (2008)
32. Sen, S., Vig, J., Riedl, J.: Tagommenders: connecting users to items through tags. In: Proceedings of 18th International World Wide Web Conference (WWW'09), pp. 671–680 (2009)
33. Steyvers, M., Griffiths, T.: Probabilistic Topic Models. Lawrence Erlbaum Associates (2007)
34. Wu, L., Yang, L., Yu, N., Hua, X.S.: Learning to tag. In: Proceedings of 18th International World Wide Web Conference (WWW'09), pp. 361–370 (2009)
35. Wu, X., Zhang, L., Yu, Y.: Exploring social annotations for the semantic web. In: Proceedings of 15th International World Wide Web Conference (WWW'06), pp. 417–426 (2006)
36. Yi, J., Maghoul, F., Pedersen, J.O.: Deciphering mobile search patterns: a study of yahoo! mobile search queries. In: Proceedings of 17th International World Wide Web Conference (WWW'08), pp. 257–266 (2008)

37. Zhang, X., Dong, G., Ramamohanarao, K.: Exploring constraints to efficiently mine emerging patterns from large high-dimensional datasets. In: Proceedings of 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'00), pp. 310–314 (2000)
38. Zhu, J., Wang, C., He, X., Bu, J., Chen, C., Shang, S., Qu, M., Lu, G.: Tag-oriented document summarization. In: Proceedings of 18th International World Wide Web Conference (WWW'09), pp. 1195–1196 (2009)