MINING DISCRIMINATIVE ITEMS IN MULTIPLE DATA STREAMS

by

Zhenhua Lin
B.Sc., Fudan University, 2008

A Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Spring 2010

APPROVAL

Name: Zhenhua Lin

Degree: Master of Science

Title of Thesis: Mining Discriminative Items in Multiple Data Streams

Examining Committee: Dr. Qianping Gu
Chair

Dr. Jian Pei, Senior Supervisor

Dr. Funda Ergun, Supervisor

Dr. Martin Ester, SFU Examiner

Date Approved: January 20th, 2010

Declaration of
Partial Copyright Licence

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

To my parent,
and my girlfriend.

"I have found a very great number of exceedingly beautiful theorems."

— Pierre de Fermat(1601 - 1665)

Acknowledgments

It is difficult to overemphasize my gratitude to my senior supervisor Dr. Jian Pei. Without his patience, his encouragement, and his suggestions, it is impossible for me to make this thesis. During my journey of graduate study toward to Master degree, he guided me to the right direction and provided lots of sound advice about research, study and life. As an advisor, he taught me many skills that I will use in my future career.

I am grateful to my supervisor, Dr. Funda Ergun, for her insightful comments and helpful suggestions helping me improve the quality of this thesis. I also thank Dr. Martin Ester and Dr. Qianping Gu for serving on my examining committee and spending their precious time on reviewing my work.

I wish to express my special thankfulness to Bin Jiang who worked closely with me, taught me countless valuable research skills without reserve, kept an eye on the progress of my work, and was always available when I needed his advise and help.

My deepest thanks go to Dr. Wei Wang and Dr. Weidong Yang at Fudan University for inspiring me to conduct research in the realm of data mining, and encouraging me to pursue further study abroad.

I specially thank some researchers important to me in Microsoft Research Asia. My intern mentors, Lei Zhang and Jin Li provided me their great help and professional training during my internship. Researcher Jiangming Yang taught me many useful skills which were used in this thesis. Researcher Dr. Daxin Jiang contributed valuable discussion to this thesis.

I would also like to acknowledge many people in our department, support staff and faculty, for always being helpful over the years. I thank my friends at Simon Fraser University for their help. A particular acknowledgement goes to Jiyi Chen, Xu Cheng, Yi Cui, Ming Hua, Luping Li, Junqiang Liu, Hossein Maserrat, Brittany Nielsen, Guanting Tang, Kate

Tsoukalas, Feng Wang, Haiyang Wang, Zhengzheng Xing, Ji Xu, Geoffrey Zenger and Bin Zhou. I also would like to express my sincere thanks to my dear friends Jie Cong, Wei Gao, Linghui Gong, Lujun Fang, Yi Han, Jin Huang, Jiarong Jiang, Shanshan Li, Yi Shen, Jiajun Wang, Jian Xu, Xiaojin Xu, Zhiting Xu, Danfeng Zhang and Zheng Zhu.

Last but not least, my dedicated thanks go to my parents, my sister, my brother and my girlfriend Liu Yang for their support, encouragement and company through all these years. I hope my achievement will make them proud of me, as I am proud of them.

Contents

List of Tables

List of Figures

Chapter 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 of a 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 [7, 8, 51]. 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 "Hollywood 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 Chapter 3 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 [25].

In this thesis, 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-pass 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.

1.1 Organization of the Thesis

The rest of the thesis is organized as follows. In Chapter 2, we formulate the problem of mining discriminative items over data streams, and give a space lower bound. In Chapter 3 we review the related work. In Chapter 4, we develop three heuristic algorithms, namely the frequent item based method in Section 4.1 which derives discriminative items from frequent items in a single stream, the hash-based method in Section 4.2 and the hybrid method in Section 4.3 combining the advantages of the frequent item based method and the hash-based method, which consumes the least space to achieve high precision and recall. Chapter 5 reports extensive experiments on real and synthetic data sets and shows that our methods are efficient and scalable. Chapter 6 concludes the thesis.

Chapter 2

Problem Definition

In this chapter, we first formulate the problem of mining discriminative items from streams. Then, we give a space lower bound and 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 minimum threshold $0 < \phi \ll 1$, and require that any discriminative item should have a frequency in S_1 no less than $\phi\theta n_1$. 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.

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

Table 2.1: A running example. x and z are discriminative items when $\theta = 3$ and $\phi = 0.1$.

Given two streams for mining discriminative items, at any query time, $\frac{n_1}{n_2}$ is the same for all items. Without loss of generality, in the rest of the thesis, 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 θ, and a minimum threshold ϕ, an item e is **discriminative** 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 thesis can be extended in the same way in the cases where $n_1 \neq n_2$.

Example 1 (Discriminative items)**.** *Table 2.1 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.* □

2.2 Space Lower Bound

Let E denote the set of discriminative items, we establish the fact that any one-pass 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-pass 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.* [30] (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. □

Note that this lower bound holds for any value of θ. 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 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 $|E| \leq \min\{|\Sigma|, \frac{1}{\phi\theta}\}$. □

For intended applications, we expect $\frac{1}{\phi\theta} \ll |\Sigma|$, because ϕ is set to 10^{-7} and θ is set to 100 or larger in general, while the alphabet of streams such as tags in Web and search queries of a search engine are far larger than 10^{-5}.

2.3 Summary of Results

The lower bound clearly indicates that any exact one-pass 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. However, for intended applications, the number of discriminative items is far less than alphabet size. This provides possibility of reducing space usage while achieving good effectiveness at the same time. In

this thesis, we develop heuristic algorithms to tackle the space limitation. Specifically, we explore three approaches.

A frequent item based method (Section 4.1) 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 4.2) 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.

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 4.3) boosts the recall of the hash-based method, and consumes the least space among the three to achieve high precision and recall.

Chapter 3

Related Work

Our problem of finding discriminative items between two streams can provide solutions to many Web mining applications such as tag suggestions [22, 28, 42, 53] summarizing web documents [40, 59], emails [10], or Web search results [32], search engine query analysis [56], social network analysis [49, 54], and so on. An essential issue inherent in those applications is to find discriminative tags or keywords that can distinguish the targeting object from many others. Statistically, we model such discrimination as frequency ratio. Due to the large amount of data arriving or being generated in high speed on the Web, the streaming model is preferred. In Section 3.1, we review different data stream models briefly. Since our work highly related to finding frequent items in data stream, we dedicate Section 3.2 to several algorithms on this problem. Finally, we review works on emerging pattern mining and tell how our work is different.

3.1 Data Stream Model

A *sequence*, or *stream*, is an ordered list of objects. A data stream $x_1, x_2, \ldots$ drawing from a discrete and ordered alphabet Σ arrives item by item. It describes an underlying signal $\mathbf{F}$ which is a one dimensional function $\mathbf{F} : \Sigma \mapsto Z$ when Z is the set of integers. Without loss of generality, assume $\Sigma = [1 \ldots M]$. There are three models varying on how items describe the signal [46].

Turnstile Model In this model, $x_i = (j, I_i)$ updates signal $\mathbf{F}_{i-1}$ by $\mathbf{F}_i[j] = \mathbf{F}_{i-1}[j] + I_i$, where $I_i \in Z$ and $\mathbf{F}_i$ denote the state of $\mathbf{F}$ after observing item x_i. Item (j, I_i) can be viewed

as inserting (when $I_i > 0$) or deleting (when $I_i < 0$) $|I_i|$ instances of j from the sequence. Note that $\mathbf{F}_i[j]$ may be negative. If we only allow that $\mathbf{F}_i[j] \geq 0$ for all $j \in [1 \ldots M]$ at all times, then we get *strict* Turnstile model. Otherwise, it is termed *non-strict* Turnstile model. To understand this model, image the following scenario. Suppose there is a bank with M different accounts providing basic service like depositing and drawing cash. A transaction $t_i = (j, I_i)$ records how much cash is deposited if $I_i > 0$ or drawn if $I_i < 0$ from account j. Also, as transaction t_i is completed, balance $\mathbf{F}_i[j]$ of account j is updated by $\mathbf{F}_i[j] = \mathbf{F}_{i-1}[j] + I_i$. As owners of accounts come without particular order to deposit or draw cash, a sequence of transactions $t_1, t_2, \ldots$, is generated. If the bank doesn't allow overdrawing, then the sequence of transactions follows strict Turnstile model. Otherwise, it fits non-strict Turnstile model.

Cash Register Model If $I_i \geq 0$ is required at all times in Turnstile model, then the new model is called Cash Register model. Clearly, Cash Register model is the special case of the Turnstile model. Use the scenario above. If the bank only provides deposit service, then the sequence of transactions fits Cash Register model. In this case, the balance of each account keeps growing as transactions commit.

Time Series Model This is the simplest one among three models and the special case of the Turnstile model, in which each $x_i = (i, I_i)$ and items show up in the increasing order of Σ. In the example of bank transaction, assume each owner of accounts comes only once in the increasing order of account number, then the transaction stream follows the Time Series model. A better example is that the bank records the total balance of all accounts every day by the form $r_i = (i, B_i)$ where B_i is the total balance of all accounts in day i.

Among three models, the Turnstile model is the most general data stream model. The Cash Register model and the Time Series model are the special cases of the Turnstile model. Ideally, researchers would like to design algorithms for the Turnstile model. However, it's hard to develop efficient algorithms on this model. From a practical point of view, although other models are weaker, they are more suitable for many applications. In this thesis, we use the Cash Register model which is the popular data stream model. Gilbert *et al.* [26] give a conventional description of this model.

3.2 Finding Frequent Item in Data Stream

Items frequent in a data stream are generally more interesting. The problem of finding frequent items which can date back to the 1980s is heavily studied in data stream research due to its intuitive interest and value. In the literature there are plenty of works on this problem with many different formulations, like finding top-k most frequent items [12, 45, 47], finding all items with frequency exceeding a user specific threshold [5, 30, 33], finding frequent items over sliding windows [5, 18, 33, 47], and so on. In this section we only review the most important algorithms in the framework of finding items whose frequencies are larger than a threshold since this formulation is most related to our problem.

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 finding the exact set E must use $\Omega(|\Sigma| \log \frac{n}{|\Sigma|})$ space in the worst case (Karp *et al.* [30]). To break down this lower bound, ϵ-approximate frequent items problem [43, 45] is introduced, whose goal is to find a set of items E so that for each item $e \in E$, $f(e) > (\phi - \epsilon)n$. Also note that if an algorithm of estimating the frequency of each item can make that for each item $e \in \Sigma$, $\hat{f}(e) \leq f(e) \leq \hat{f}(e) + \epsilon n$ where $\hat{f}(e)$ is the estimated frequency of e, then it can be used to solve the ϵ-approximate frequent item problem by simply reporting those e whose frequency is estimated above $(\phi - \epsilon)n$.

Recently, Cormode and Hadjieleftheriou [14] compared several algorithms on this subject, and divided them into three classes, namely counter-based algorithms [9, 19, 30, 43, 45], quantile algorithms [27, 43], and randomized sketch algorithms [4, 16, 17]. Here we only review counter-based algorithms since they are more specific on finding frequent items and at the same time estimating frequent items' frequencies and therefore more related to our work. Algorithms in the other two classes solve more general problems like estimating frequency for all items. Detailed description and comparison of these algorithms can be found in [14].

The *Frequent Algorithm* was discovered independently by Karp *et al.* [30] and Demaine *et al.* [19], which generalized Majority algorithm [24, 9] of finding the item whose frequency exceeds $n/2$. The central idea of these algorithms is "Cancelation". They maintains $\frac{1}{\phi} - 1$ (item, counter) pairs. If the new item $x_i = (j, I_i)$ is among these pairs, increment the corresponding counter by I_i. Else, allocate a counter of I_i for this item if there is some counter with zero; otherwise, decrement all counters by I_i. 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 counter 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 [14]. Space used by Frequent Algorithm is $O(1/\phi)$, which is independent from the size of stream.

The *Lossy Counting* algorithm proposed by Manku and Motwani [43] stores tuples (j, l_j, δ_j) where j is the element from Σ, l_j is the lower bound of j's counter and δ_j satisfies $l_j \leq f(j) \leq l_j + \delta_j$. When an item $x_i = (j, I_i)$ arrives, if j is stored, then increment its lower bound by I_i. Otherwise, create a new tuple $(j, I_i, \lfloor \phi i \rfloor)$. From time to time, the algorithm deletes tuples with $l_j + \delta_j < \phi i$. Like the frequent algorithm, when the algorithm terminates, all items with frequency larger than ϕn are stored and the error on 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 frequency. In terms of space usage, it requires $O(\frac{1}{\phi} \log \phi n)$ in the worst case.

The *Space-Saving algorithm* [45] employed in this thesis is also a one-pass counter-based algorithm similar to Lossy Counting 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 [14] indicate that the space-saving algorithm outperforms other algorithms in terms of precision, recall, and space usage. The more detailed description can be found in Section 4.1.1.

3.3 Emerging Pattern Mining

Emerging patterns (EPs) were introduced in [20]. EPs are defined as itemsets whose supports in one dataset are significantly larger than their supports in another. Formally, let $supp_i(X)$ denote the support of X in dataset i 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 EPs with growth rate more than ϕ. These EPs are called ϕ-EPs. In addition to introducing EPs, Dong and Li [20] proposed a border-based method to mine EPs. The border is a structure to concisely represent a large collection

of itemsets. An ordered pair $\langle L, R\rangle$ of two antichain collections (A collection of sets is called antichain if for any two elements A and B in this collection, $A \nsubseteq B$ and $B \nsubseteq A$) of sets is called a *border* if each element in L is a subset of some element in R and each element in R is a superset of some element in L. Then, a border $\langle L, R\rangle$ can represent a collection $\{X|\exists Y \in L, \exists Z \in R : Y \subseteq X \subseteq Z\}$. Algorithms of mining EPs manipulate borders representing EPs, instead of EPs themselves. The use of borders avoids handling exponentially many candidates, which improves efficiency. Works by Zhang *et al* [58] was also based on borders.

Two more efficient algorithms based on tree structure inspired by FP-tree [29] were proposed by Bailey *et al.* [6] to discover *Jumping Emerging Patterns* (JEPs) which are EPs with infinite growth rate, and by Fan and Ramamohanarao [23] to mine *essential JEPs* (eJEPs) which are JEPs and whose subsets are not JEPs. Also, mining EPs in data streams is studied in [13] recently, in which the *EFI-Mine* algorithm based on A priori algorithm [50] was proposed to find EPs from a sliding window.

The applications of EPs include analyzing biological data [36, 39, 55] and building classifiers [21, 34, 35, 37, 38]. The central idea is to perform classification by leveraging the power of EPs present in instances to be classified. Basically, classifiers using EPs first select a set of EPs for each class. Then they aggregate the power of discriminativeness of EPs from each testing tuple to make a classification decision. Li *et al.* [37] showed that classifiers built in this way can outperform C5.0 [48] and Li *et al.* [36] and Yeoh *et al.* [55] showed that it can beat Support Vector Machine [52] in some datasets.

Although our problem can be viewed as a special case of emerging pattern mining by considering EPs containing only one item, the essential difference of item and itemset leads to very different solutions and applications. Most effort is devoted to addressing problem of compactly representing EPs and efficient manipulation of EPs when studying EP mining. However, in our problem, we don't have such combinatorical nature inherent in EP mining. Furthermore, we focus on reducing space of use under the data stream setting. To the best of our knowledge, we are the first to study the problem of finding discriminative items between two data streams.

Chapter 4

Methods

4.1 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 [45] is the state-of-the-art method with low space complexity and high accuracy [14]. In this section, we first briefly review the space-saving algorithm, and then show how to extend it to find discriminative items.

4.1.1 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 the frequency 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 [45]). *Assuming $\phi = 0.3$, let us find frequent items in S_1 in Table 2.1 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 [45].

4.1.2 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 2.1. 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.* □

4.1.3 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 [45].

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.

4.2 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.

4.2.1 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*

$$\sum_{e\in T} f_1(e) \geq \theta \sum_{e\in T} f_2(e), \tag{4.1}$$

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 every 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 (4.1), 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.

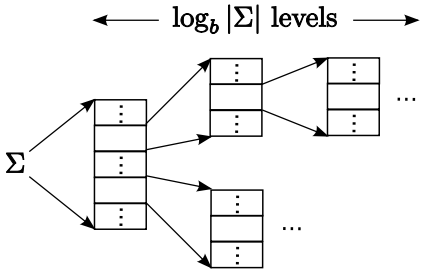


Figure 4.1: An illustration of the hierarchical hashing.

4.2.2 Hierarchical Hashing

Figure 4.1 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 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*

$$C_1(B) \geq \theta(C_2(B) + \sum_{B' \in Anc(B)} C_2(B')), \tag{4.2}$$

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 $\sum_{e \in B} f_2(e) \leq C_2(B) + \sum_{B' \in Anc(B)} C_2(B')$ due to the construction of the hierarchical hashing. 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 (4.2) 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 is 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.

$B_{1,1}$	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

Table 4.1: A running example of the hash-based method.

A bucket that does not satisfy Formula (4.2) 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 2.1. 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 4.1 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 (4.2) and* $C_1(B_{2,1})$ *is larger than the minimum frequency support* 3*. However, the hash-bashed 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.

4.2.3 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

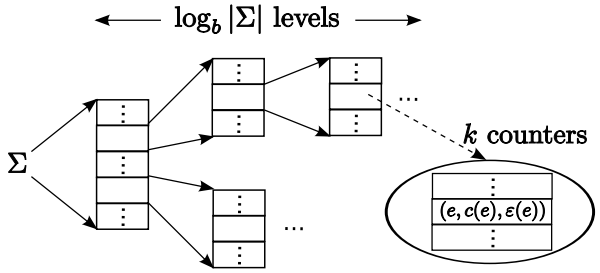


Figure 4.2: An illustration of the hybrid method.

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.

4.3 Hybrid Method

4.3.1 The Method

To discover the concealed discriminative items in a non-discriminative bucket B (i.e., a bucket that does not satisfy Formula (4.2)), we need a more aggressive expanding criterion 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 4.2 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 counter of an item $e \in B$ is 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 4.1.1, 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. To utilize this advantage of frequent items in S_1, we expand B to the next level if all elements kept in the summary of B have frequency in S_1 guaranteed to satisfy minimum support in S_1 required by discriminative items. However, at the beginning, n is small so that we can expect too many buckets will be expanded. So we use μ to control the number of expanding buckets in the early stage of HY algorithm. As n becomes large, $\phi\theta n$ will dominate μ.

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 improved. Essentially, the hybrid method trades precision for recall. Our experiments in Chapter 5 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 2.1, 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.* □

4.3.2 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 $O(\frac{hb\log_b|\Sigma|}{\phi\theta})+O(\frac{hbk}{\phi\theta}) = O(\frac{hb(\log_b|\Sigma|+k)}{\phi\theta})$, 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 [45].

Algorithm 1 The hash-based method.

Input: two streams S_1 and S_2; parameters ϕ and θ;
Output: the set E of discriminative items;
Description:

construct h independent hierarchical hashing structures and initialize their first level buckets;
for all item $e \in S_i$ $(i = 1, 2)$ **do**
  for all hierarchical hashing H **do**
    /* hashing items */
    let B be the bucket on the first level of H where e is hashed into;
    while B is not a leaf bucket **do**
      assume e is hashed into the child bucket B' of B;
      $B = B'$;
    end while
    $C_i(B) = C_i(B) + 1$;
    /* growing hashing */
    if B is discriminative (Lemma 2) and B is not on the $\log_b |\Sigma|$-th level **then**
      apply a uniform hash with b buckets on the items of B to construct the next level hashing;
    end if
    /* deleting buckets */
    if e is from stream S_2 **then**
      let B_p be the parent of B;
      while non of B_p's child bucket is discriminative **do**
        $C_i(B_p) = \sum_{B_c \in Chi(B_p)} C_i(B_c)$;
        delete all child buckets of B_p;
        assign B_p the parent of B_p;
      end while
    end if
  end for
end for
return all items in the discriminative buckets on the $\log_b |\Sigma|$ level;

Chapter 5

Empirical Studies

We conducted experiments on real and synthetic data sets to evaluate the accuracy and efficiency of our three methods, 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 minimum threshold is $\phi = 10^{-6}$ and does not change 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 implemented by the method stated in [11].

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.

5.1 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, 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 $\frac{n_2}{n_1}$.

5.1.1 Efficiency

Figure 5.1 compares the space usage in the three methods. FE uses the most space among the three. It is only dependent from the minimum threshold ϕ and 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 5.1(a) shows that its space usage is not sensitive 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 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 5.1(c) 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 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 5.2. 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 5.2(b), 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 5.2(a) shows

that HY is slightly faster than HA, since HY uses less hierarchical hashing than HA. When using the same number of hashing, Figure 5.2(c) shows that HY is slower than HA, as HY needs to maintain the space-saving summary.

5.1.2 Accuracy

Figure 5.3 compares the precision of the three methods. As stated in Sections 4.1 and 4.2, 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 5.4 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 5.4(a), 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 5.4(e) 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 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.

5.2 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 5.1. We divide the 20 newsgroups into to 2 partitions as shown in Table 5.1, 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. To test whether algorithms are sensitive to burst, we use two

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

Table 5.1: Topics in real data sets.

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

Table 5.2: Size of the real data sets in words

mathematics against law		law against mathematics	
words	ratio	words	ratio
polynomial	855.55	constitution	1010.25
algebra	761.14	jurisdiction	894.75
algebraic	703.65	justice	480.38
geometry	679.17	parliament	448.58
topology	645.50	defendant	442.86

Table 5.3: Top-5 most discriminative words in the Wikipedia data set

ways to merge articles: 1) merge articles randomly; 2) merge articles according to publishing time order. We use " 20 Newsgroup (random)" to refer to data streams generated by first way, and "20 Newsgroup (ordered)" to refere to data streams generated by second way.

For all articles, we only conduct stemming but do not filter out stopping words. Table 5.2 lists the size of the two partitions of each data set.

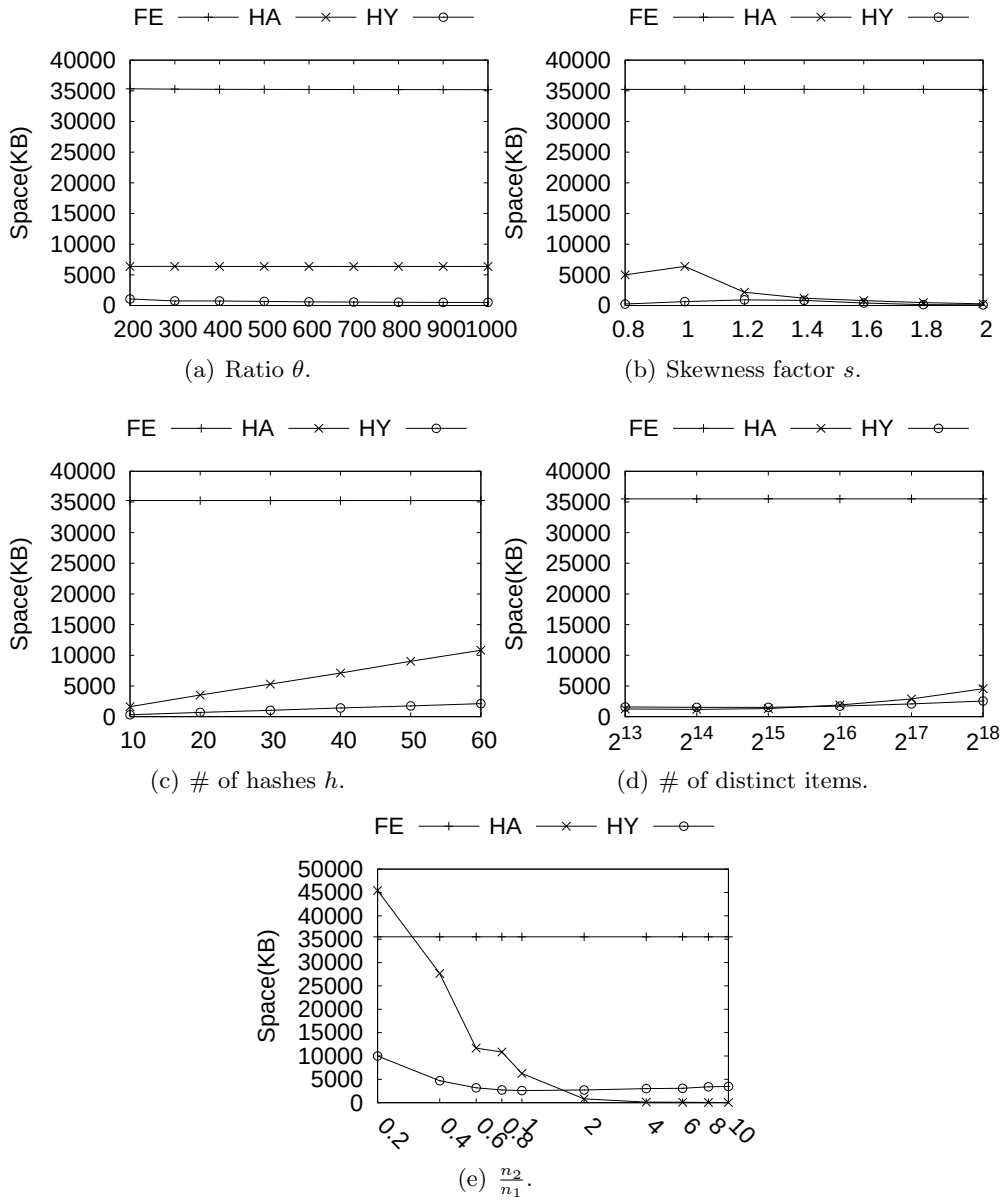
Table 5.3 lists the top-5 high ratio words in the Wikipedia data set, which match our common intuition. Figures 5.5 and 5.6 show the ratio distribution of all terms on the two real data sets in log-log graph. We observe a power law distribution of the ratio. The sharp tails are caused by the minimum threshold ϕ.

Tables 5.4, 5.5 and 5.6 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 both the Wikipedia data set and the 20 Newsgroup (random), 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.

In bursting data (20 Newsgroup (ordered)), in some cases FE may not achieve a recall of 100%. This is because bursts of data streams will result in a relatively low precision of frequency estimation for some frequent items by Space Saving algorithm. HA are relatively sensitive to bursts since bursts from S_2 will reduce the number of expanding buckets. However, bursts have little impact on HY algorithm.



(a) Ratio θ.

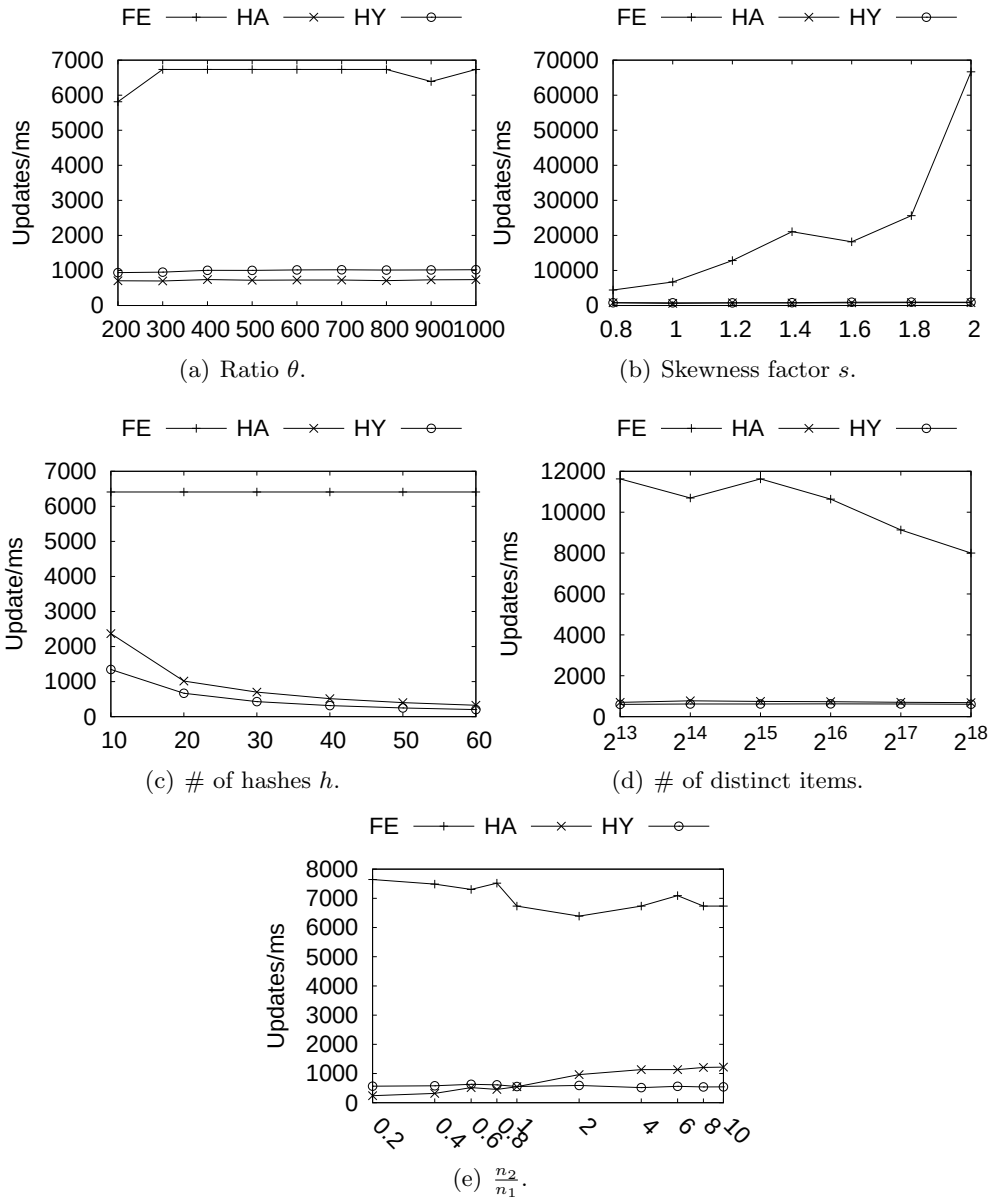
(b) Skewness factor s.

(c) # of hashes h.

(d) # of distinct items.

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

Figure 5.1: Space on synthetic data sets.



(a) Ratio θ.

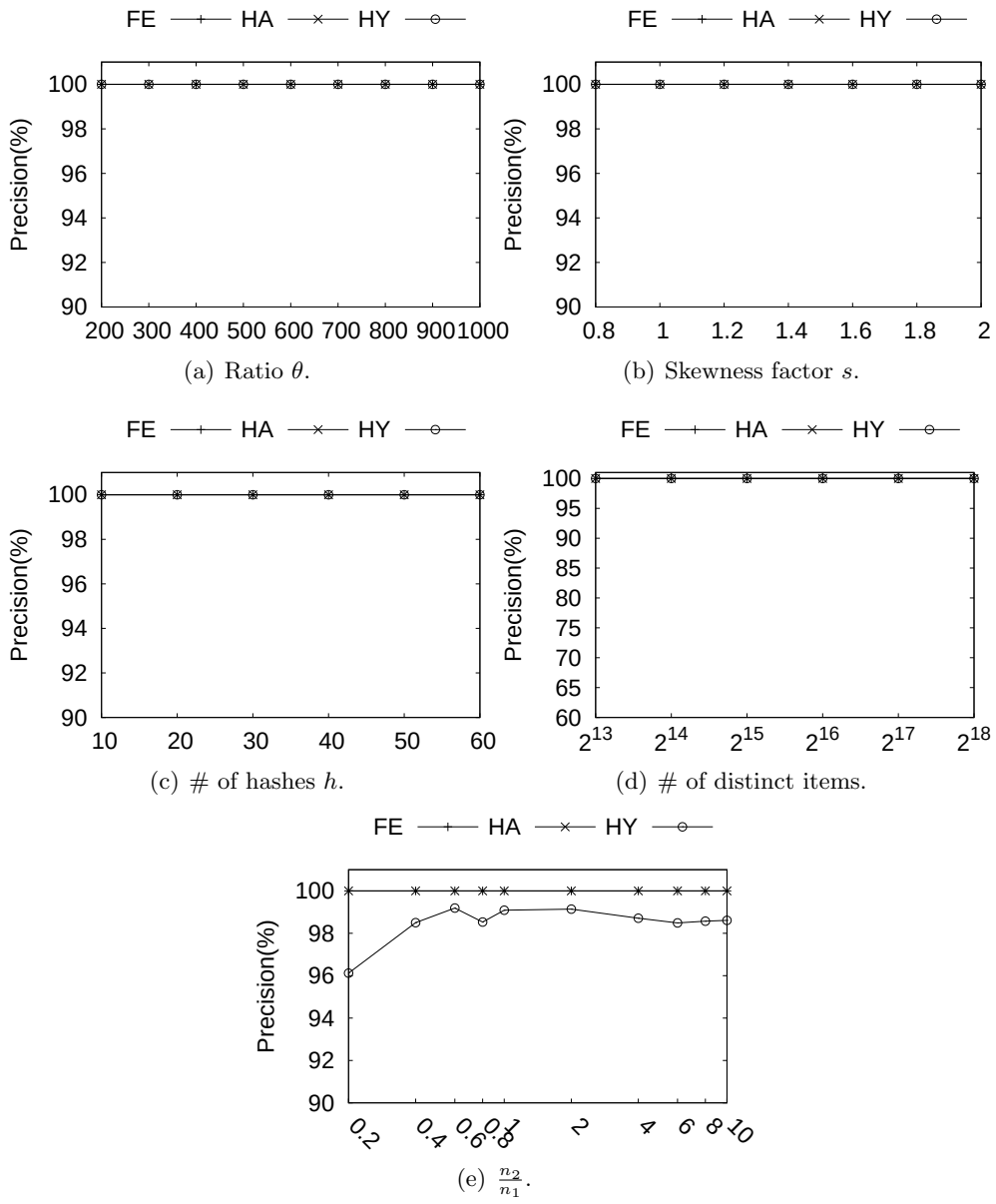
(b) Skewness factor s.

(c) # of hashes h.

(d) # of distinct items.

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

Figure 5.2: Number of updates per ms on synthetic data sets.



(a) Ratio θ.

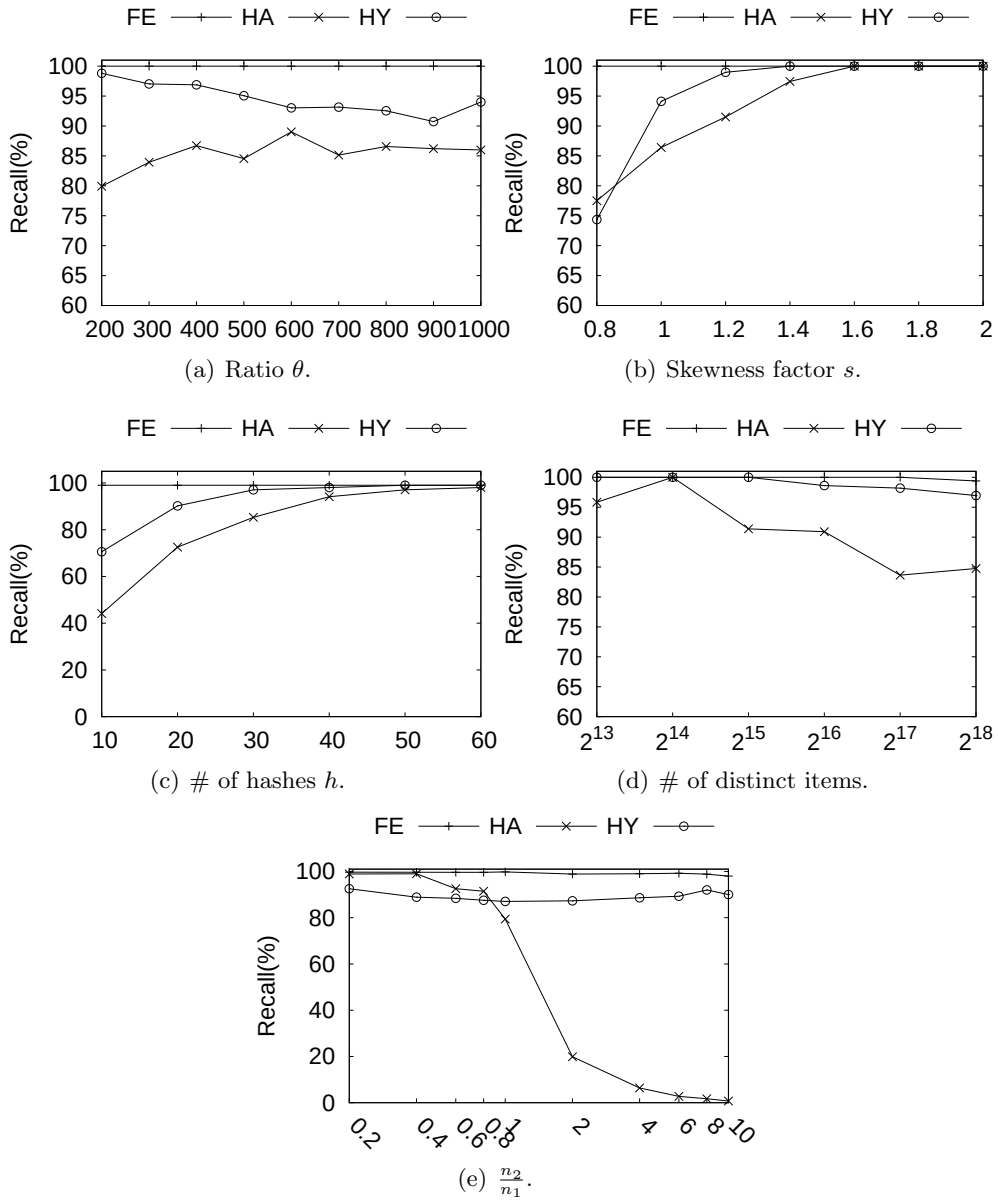
(b) Skewness factor s.

(c) # of hashes h.

(d) # of distinct items.

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

Figure 5.3: Precision on synthetic data sets.



(a) Ratio θ.

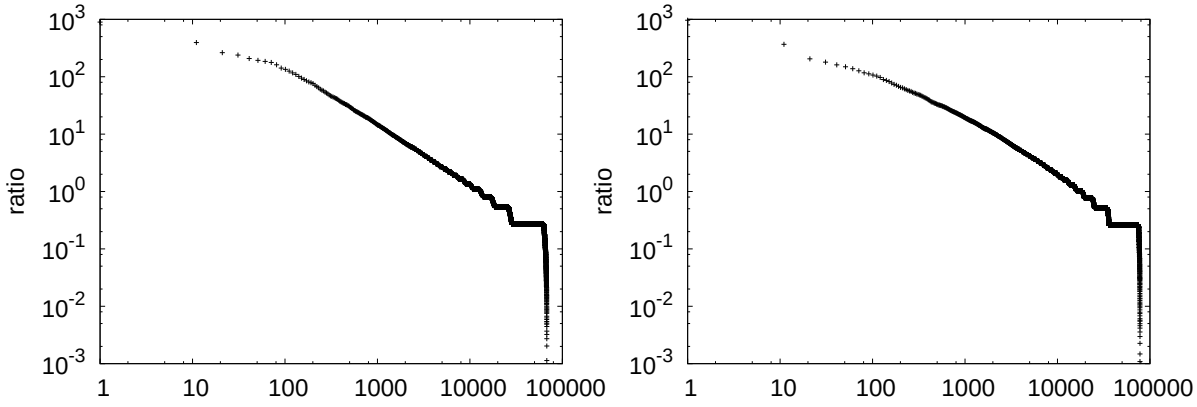
(b) Skewness factor s.

(c) # of hashes h.

(d) # of distinct items.

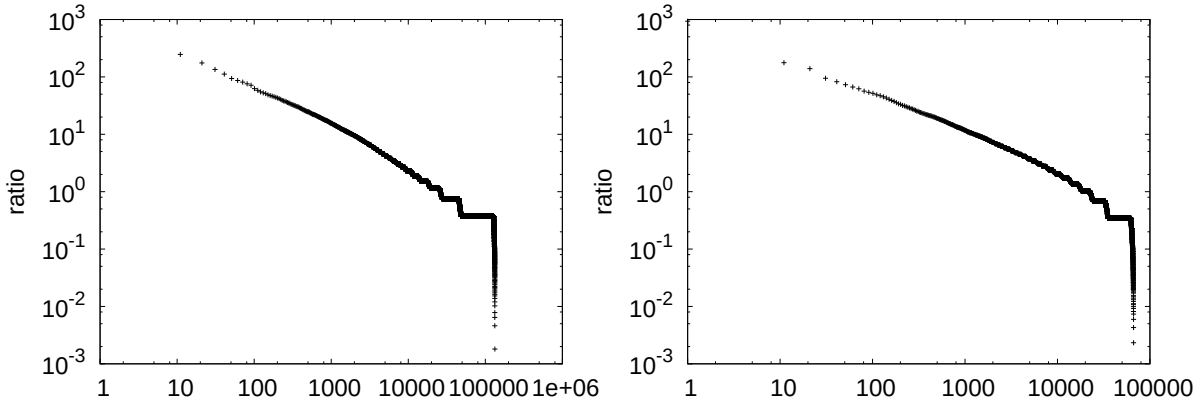
(e) $\frac{n_2}{n_1}$.

Figure 5.4: Recall on synthetic data sets.



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

Figure 5.5: The distribution of ratio on the Wikipedia data set.



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

Figure 5.6: The distribution of ratio on the Newsgroups data set.

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

θ		space(MB)	updates/ms	precision(%)	recall(%)
100	FE	34.68	20967.74	100.00	100.00
	HA	0.97	1068.17	100.00	56.64
	HY	0.29	5803.71	100.00	89.51
300	FE	34.44	20967.74	100.00	100.00
	HA	0.96	1063.50	100.00	61.11
	HY	0.22	5803.71	100.00	83.33

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

θ		space(MB)	updates/ms	precision(%)	recall(%)
100	FE	34.68	20073.11	100.00	100.00
	HA	1.72	1082.61	100.00	59.29
	HY	0.50	5668.24	97.87	81.42
300	FE	34.44	19301.07	100.00	100.00
	HA	1.71	1087.46	100.00	69.23
	HY	0.51	5732.99	100.00	84.62

Table 5.4: Results on the Wikipedia data set.

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

θ		space(MB)	updates/ms	precision(%)	recall(%)
100	FE	34.68	17795.71	100.00	100.00
	HA	3.24	981.49	100.00	59.57
	HY	0.65	5470.21	91.67	93.62
200	FE	34.50	16927.63	100.00	100.00
	HA	3.23	973.57	100.00	68.75
	HY	0.36	5552.26	93.75	93.75

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

θ		space(MB)	updates/ms	precision(%)	recall(%)
100	FE	34.68	16876.18	100.00	100.00
	HA	1.12	984.27	100.00	33.33
	HY	0.48	5228.12	82.35	93.33
200	FE	34.50	16140.30	100.00	100.00
	HA	1.11	989.71	100.00	42.86
	HY	0.30	5464.82	77.78	100.00

Table 5.5: Results on the Newsgroups data set (random).

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

θ		space(MB)	updates/ms	precision(%)	recall(%)
100	FE	34.68	17738.86	100.00	100.00
	HA	0.98	1045.23	100.00	14.89
	HY	0.53	5464.82	95.74	95.74
200	FE	34.50	16927.63	100.00	87.50
	HA	0.98	1045.23	100.00	25.00
	HY	0.31	5464.82	93.75	93.75

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

θ		space(MB)	updates/ms	precision(%)	recall(%)
100	FE	34.68	16927.63	100.00	100.00
	HA	2.24	939.95	100	70.00
	HY	0.45	5552.26	92.86	86.67
200	FE	34.50	16187.35	100.00	100.00
	HA	2.25	937.57	100.00	85.71
	HY	0.38	5636.81	100.00	100.00

Table 5.6: Results on the Newsgroups data set (ordered).

Chapter 6

Discussions and Conclusions

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. In this chapter, we first summarize the thesis and then discuss future work.

6.1 Summary of the Thesis

First, *we study the discriminative items mining problem.* An item is discriminative if it is frequent in one stream while relatively infrequent in another. We formulate this problem as the ratio of frequency rates of an item in two streams. From minimum support point of view, we introduce minimum threshold of frequency rate in S_1 so that insignificant and less interesting items are excluded. Under this definition, we show that at least space linear to size of the alphabet is required for any exact online algorithm in the worst case. To cut down space of use, we develop three heuristic algorithms which in practice can achieve good performance in terms of precision, recall and space.

Second, *we evaluate our algorithms in empirical setting.* We conduct extensive experiment on both synthetic data following Zipf distribution and real data extracting from 20 newsgroup dataset and Wikipedia. The experimental results on these dataset are encouraging.

6.2 Future Work

Finding discriminative items from one stream against multiple Streams. In this thesis, we only consider two streams. However, in practice, we may have more than two streams, say $k > 2$ streams $S_1, S_2, \ldots, S_k$, and we are interested in discriminative items whose frequency is large in stream S_1 but relatively small in all other streams. Straightforwardly, we can devise algorithms proposed in this thesis to all pairs (S_1, S_i) $(i \neq 1)$ of streams and then assemble discriminative items found in each pair, for example, outputting items discriminative in all pairs. However, this naive extension is not efficient since it has to maintain $k - 1$ running copies of algorithms for S_1 against others. Also, note that we can not simply combine streams $S_2, \ldots, S_k$ together as one stream S' and then use algorithms presented in this thesis on S_1 and S' to solve this problem, because an item x is discriminative in S_1 against S' may not be a discriminative item in S_1 against all other streams. For example, suppose x has comparable frequencies in S_1 and S_2 so that x is *not* a discriminative item in S_1 against S_2, while it vanishes in other streams so that its frequency is small enough in S' to make it become a discriminative item in S_1 against S'. This case indicates the problem of finding discriminative items from one stream against multiple streams is far from trivial. As a future direction, we would like to explore efficient algorithms and data structures on one stream against multiple streams problem.

Finding top-k discriminative items. Discovering all discriminative items given threshold θ is meaningful, but sometimes is unnecessary. For example, users may be only interested in top-k discriminative items in tag streams. Moreover, although we know that the number of discriminative items is at most $\frac{1}{\theta\phi}$, we can not predict the number of discriminative items exactly. In some cases, for example, when the threshold θ is small, there are too many discriminative items, while in some cases when θ is large, there are too few discriminative items, even none. In these cases, instead of finding all discriminative items, outputting top-k discriminative items is more suitable. Also, to specify how many items should be returned (the k value) is easier than to set up an appropriate threshold θ for users without knowledge of the domain and the streams in question.

Bibliography

[1] Morgan Kaufmann, 2002.

[2] *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, volume 148 of *ACM International Conference Proceeding Series*. ACM, 2006.

[3] *Computer Science - Theory and Applications, Third International Computer Science Symposium in Russia, CSR 2008, Moscow, Russia, June 7-12, 2008, Proceedings*, volume 5010 of *Lecture Notes in Computer Science*. Springer, 2008.

[4] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC'96)*, pages 20–29, 1996.

[5] Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of 23th ACM SIGMOD Principles of Database Systems (PODS'04)*, pages 286–296, 2004.

[6] James Bailey, Thomas Manoukian, and Kotagiri Ramamohanarao. Fast algorithms for mining emerging patterns. In *Proceedings of 6th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'02)*, pages 39–50, 2002.

[7] David M. Blei and John D. Lafferty. Dynamic topic models. In *Proceedings of the 23rd International Conference (ICML'06)* [2], pages 113–120.

[8] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[9] Robert S. Boyer and J. Strother Moore. Mjrty: A fast majority vote algorithm. In *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 105–118, 1991.

[10] Giuseppe Carenini, Raymond T. Ng, and Xiaodong Zhou. Summarizing email conversations with clue words. In *Proceedings of 16th International World Wide Web Conference (WWW'07)*, pages 91–100, 2007.

[11] Larry Carter and Mark N. Wegman. Universal classes of hash functions (extended abstract). In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing (STOC'77)*, pages 106–112, 1977.

[12] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Proceedings of 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, pages 693–703, 2002.

[13] Chun-Jung Chu, Vincent S. Tseng, and Tyne Liang. Efficient mining of temporal emerging itemsets from data streams. *Expert Systems with Applications*, 36(1):885–893, 2009.

[14] Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment (PVLDB'08)*, 1(2):1530–1541, 2008.

[15] Graham Cormode and Marios Hadjieleftheriou. Finding the frequent items in streams of data. *Communications of the ACM*, 52(10):97–105, 2009.

[16] Graham Cormode and S. Muthukrishnan. 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.

[17] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[18] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows (extended abstract). In *Proceedings of 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'02)*, pages 635–644, 2002.

[19] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Frequency estimation of internet packet streams with limited space. In *Proceedings of 10th European Symposium on Algorithms (ESA'02)*, pages 348–360, 2002.

[20] Guozhu Dong and Jinyan Li. 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)*, pages 43–52, 1999.

[21] Guozhu Dong, Xiuzhen Zhang, Limsoon Wong, and Jinyan Li. Caep: Classification by aggregating emerging patterns. In *Proceedings of 2nd International Conference on Discovery Science (DS'99)*, pages 30–42, 1999.

[22] Micah Dubinko, Ravi Kumar, Joseph Magnani, Jasmine Novak, Prabhakar Raghavan, and Andrew Tomkins. Visualizing tags over time. In *Proceedings of 15th International World Wide Web Conference (WWW'06)*, pages 193–202, 2006.

[23] Hongjian Fan and Kotagiri Ramamohanarao. 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), pages 456–462, 2002.

[24] Michael J. Fischer and Steven L. Salzberg. Finding a majority among n votes. In *Journal of Algorithms*, volume 3, pages 376–379, 1982.

[25] Sumit Ganguly. Lower bounds on frequency estimation of data streams. In *Proceedings of 3rd International Computer Science Symposium in Russia (CSR'08)* [3], pages 204–215.

[26] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proceedings of 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 79–88, 2001.

[27] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data (SIGMOD'01)*, pages 58–66, 2001.

[28] Harry Halpin, Valentin Robu, and Hana Shepherd. The complex dynamics of collaborative tagging. In *Proceedings of 16th International World Wide Web Conference (WWW'07)*, pages 211–220, 2007.

[29] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data (SIGMOD'00)*, pages 1–12, 2000.

[30] Richard M. Karp, Scott Shenker, and Christos H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transaction of Database Systems*, 28:51–55, 2003.

[31] Raymond Kosala and Hendrik Blockeel. Web mining research: A survey. *SIGKDD Explorations*, 2(1):1–15, 2000.

[32] Byron Y.-L. Kuo, Thomas Hentrich, Benjamin M. Good, and Mark D. Wilkinson. Tag clouds for summarizing web search results. In *Proceedings of 16th International World Wide Web Conference (WWW'07)*, pages 1203–1204, 2007.

[33] Lap-Kei Lee and H. F. Ting. 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)*, pages 290–297, 2006.

[34] Jinyan Li, Guozhu Dong, and Kotagiri Ramamohanarao. Making use of the most expressive jumping emerging patterns for classification. *Knowledge and Information Systems*, 3(2):131–145, 2001.

[35] Jinyan Li, Guozhu Dong, Kotagiri Ramamohanarao, and Limsoon Wong. Deeps: A new instance-based lazy discovery and classification system. *Machine Learning*, 54(2):99–124, 2004.

[36] Jinyan Li, Huiqing Liu, James R. Downing, Allen Eng-Juh Yeoh, and Limsoon Wong. Simple rules underlying gene expression profiles of more than six subtypes of acute lymphoblastic leukemia (all) patients. *Bioinformatics*, 19(1):71–78, 2003.

[37] Jinyan Li, Kotagiri Ramamohanarao, and Guozhu Dong. Combining the strength of pattern frequency and distance for classification. In *Proceedings of 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'01)*, pages 455–466, 2001.

[38] Jinyan Li and Limsoon Wong. Geography of differences between two classes of data. In *Proceedings of 6th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'02)*, pages 325–337, 2002.

[39] Jinyan Li and Limsoon Wong. Identifying good diagnostic gene groups from gene expression profiles using the concept of emerging patterns. *Bioinformatics*, 18(10):1406–1407, 2002.

[40] Liangda Li, Ke Zhou, Gui-Rong Xue, Hongyuan Zha, and Yong Yu. Enhancing diversity, coverage and balance for summarization through structure learning. In *Proceedings of 18th International World Wide Web Conference (WWW'09)*, pages 71–80, 2009.

[41] Bing Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Data-Centric Systems and Applications. Springer, 2007.

[42] Dong Liu, Xian-Sheng Hua, Linjun Yang, Meng Wang, and Hong-Jiang Zhang. Tag ranking. In *Proceedings of 18th International World Wide Web Conference (WWW'09)*, pages 351–360, 2009.

[43] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02)* [1], pages 346–357.

[44] Benjamin Markines, Ciro Cattuto, Filippo Menczer, Dominik Benz, Andreas Hotho, and Gerd Stumme. Evaluating similarity measures for emergent semantics of social tagging. In *Proceedings of 18th International World Wide Web Conference (WWW'09)*, pages 641–650, 2009.

[45] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *Proceedings of 10th International Conference on Database Theory (ICDT'05)*, pages 398–412, 2005.

[46] S. Muthukrishnan. *Data streams: algorithms and applications*. Now Publishers Inc., 2005.

[47] Shu Pingda and Chen Huahui. 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)*, volume 4, pages 267–270, 2008.

[48] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[49] Shilad Sen, Jesse Vig, and John Riedl. Tagommenders: connecting users to items through tags. In *Proceedings of 18th International World Wide Web Conference (WWW'09)*, pages 671–680, 2009.

[50] Ramakrishnan Srikant and Rakesh Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data (SIGMOD'96)*, pages 1–12, 1996.

[51] Mark Steyvers and Tom Griffiths. *Probabilistic Topic Models*. Lawrence Erlbaum Associates, 2007.

[52] Vladimir Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[53] Lei Wu, Linjun Yang, Nenghai Yu, and Xian-Sheng Hua. Learning to tag. In *Proceedings of 18th International World Wide Web Conference (WWW'09)*, pages 361–370, 2009.

[54] Xian Wu, Lei Zhang, and Yong Yu. Exploring social annotations for the semantic web. In *Proceedings of 15th International World Wide Web Conference (WWW'06)*, pages 417–426, 2006.

[55] Eng-Juh Yeoh, Mary E Ross, Sheila A Shurtleff, W.Kent Williams, Divyen Patel, Rami Mahfouz, Fred G Behm, Susana C Raimondi, Mary V Relling, Anami Patel, Cheng Cheng, Dario Campana, Dawn Wilkins, Xiaodong Zhou, Jinyan Li, Huiqing Liu, Ching-Hon Pui, William E Evans, Clayton Naeve, Limsoon Wong, and James R Downing. Classification, subtype discovery, and prediction of outcome in pediatric acute lymphoblastic leukemia by gene expression profiling. *Cancer Cell*, pages 133–143, 2002.

[56] Jeonghee Yi, Farzin Maghoul, and Jan O. Pedersen. Deciphering mobile search patterns: a study of yahoo! mobile search queries. In *Proceedings of 17th International World Wide Web Conference (WWW'08)*, pages 257–266, 2008.

[57] Qingyu Zhang and Richard S. Segall. Web mining: a survey of current research, techniques, and software. *International Journal of Information Technology and Decision Making*, 7(4):683–720, 2008.

[58] Xiuzhen Zhang, Guozhu Dong, and Kotagiri Ramamohanarao. 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)*, pages 310–314, 2000.

[59] Junyan Zhu, Can Wang, Xiaofei He, Jiajun Bu, Chun Chen, Shujie Shang, Mingcheng Qu, and Gang Lu. Tag-oriented document summarization. In *Proceedings of 18th International World Wide Web Conference (WWW'09)*, pages 1195–1196, 2009.