Frequent Item Mining When Obtaining Support is Costly

Joe Wing-Ho Lin and Raymond Chi-Wing Wong

The Hong Kong University of Science and Technology
{whlinaa,raywong}@cse.ust.hk

Abstract. Suppose there are n users and m items, and the preference of each user for the items is revealed only upon probing, which takes time and is therefore costly. How can we quickly discover all the frequent items that are favored individually by at least a given number of users? This new problem not only has strong connections with several well-known problems, such as the frequent item mining problem, it also finds applications in fields such as sponsored search and marketing surveys. Unlike traditional frequent item mining, however, our problem assumes no prior knowledge of users' preferences, and thus obtaining the support of an item becomes costly. Although our problem can be settled naively by probing the preferences of all n users, the number of users is typically enormous, and each probing itself can also incur a prohibitive cost. We present a sampling algorithm that drastically reduces the number of users needed to probe to $O(\log m)$—regardless of the number of users—as long as slight inaccuracy in the output is permitted. For reasonably sized input, our algorithm needs to probe only 0.5% of the users, whereas the naive approach needs to probe all of them.

Keywords: frequent item mining, random sampling

1 Introduction

We propose a new data mining problem called HIDDEN FREQUENT ITEM MINING: Suppose we have a set of users $\mathcal{U}$ and a set of items $\mathcal{T}$, and that the preference of each user for the items is revealed only upon probing, which takes time and is therefore costly. Our aim is to quickly discover all the *frequent items* that are favored (i.e., the best choice) individually by at least a predefined number of users. This problem finds applications in areas such as sponsored search and marketing surveys.

Connections with Existing Problems. Our problem has strong connections with several well-recognized problems, including frequent item mining [5, 12, 13, 20], heavy hitter finding [7, 8, 17], and estimation of population proportion problem in survey sampling [21, 22]. As will be clear in Section 3, although our problem resembles the frequent item mining problem, one crucial difference sets it apart from its counterpart: in our problem, users' preferences are unknown in advance and therefore obtaining them is costly. In contrast, our counterpart assumes that the items purchased in each transaction (which may be viewed as the preferences of a user in our problem) are already known. We will also see that existing problems, such as heavy hitter finding, can be adapted to tackle a special case of our problem. Unfortunately, such an adaption results in an excessively high cost, thus rendering it impractical. Similarly, the estimation of population proportion problem in survey sampling can be employed to tackle a special case of our problem. However, existing solutions to this problem make extra assumptions in their results and are generally less efficient than our solution. Despite sharing similarities with existing studies, our proposed problem—to the best of our knowledge—has neither been proposed before, nor can it be efficiently solved by adapting solutions to existing problems.

Item	Users favoring the item	Support
t_1	u_1,u_2,u_3	3
t_2	u_1,u_4,u_5,u_6	4
t_3	u_5	1
t_4	None	0

Table 1. Illustration. We have $n = |\mathcal{U}| = 6$, $m = |\mathcal{T}| = 4$, $\mathcal{U} = \{u_1,u_2,u_3,u_4,u_5,u_6\}$, and $\mathcal{T} = \{t_1,t_2,t_3,t_4\}$. If $p = 1/3$ so that $pn = (1/3)\cdot 6 = 2$, then both t_1 and t_2 are frequent items, since their individual support is at least $pn = 2$. However, neither t_3 nor t_4 is a frequent item, since their individual support is below 2. In this example, both users u_1 and u_5 favor two items each, whereas other users favor only one item each.

Proposed Problem. Given a set $\mathcal{U}$ of n users and a set $\mathcal{T}$ of m items, we say that a user $u \in \mathcal{U}$ *favors* item $t \in \mathcal{T}$ if user u gives item t the highest score (or ranking), relative to other items in $\mathcal{T}$. We have at our disposal a *favorite-probing query* (or *query* for short) q, which probes—and subsequently returns—the favorite items $q(u)$ of a given user $u \in \mathcal{U}$. For simplicity, we sometimes say *"probe a user"* to mean that we issue a query to discover a user's favorite items. We assume that the preference of a user remains *unknown* until a favorite-probing query is issued to probe about it. In addition, we measure the *support* of an item by the number of users who favor the item. A user can favor multiple items simultaneously by giving the highest score to multiple items.

Problem Aim. The aim of our problem is to discover all the items whose individual support is at least pn, where $p \in (0,1]$ is a parameter called *support proportion* and n is the number of users. In other words, each of these items must be favored by at least $p \times 100\%$ of all users. We call each of these items a *frequent item* (or sometimes *popular item*). Table 1 shows an example of our problem.

Worst-Case Lower Bound & Remedy. Readers might raise this question: Is it always possible to solve the proposed problem without probing the preferences of all users? The answer is no if an exact answer is always desired; for instance, if a particular item t is favored by $pn-1$ users, then we must probe all n users until we can confirm whether item t is frequent. This is because leaving any user's preference unaccounted for opens up the possibility that the user favors item t. This in turn implies that *every* exact algorithm requires at least n queries in the worst case—meaning that our problem has a lower bound of n query. This result naturally motivates us to consider another question: would it be *fatal* to return an item whose support is only marginally lower than the threshold pn? We believe the answer is no, as supported by the applications to be discussed in Section 2.

Contributions. We are the first to propose the HIDDEN FREQUENT ITEM MINING problem, which finds applications in a variety of areas. Our problem also bears a strong relationship to numerous database problems. To efficiently solve our problem, we present the first sampling algorithm that outperforms the naive approach and other competing methods both in theory and in practice. Our analysis shows that to fulfil a strong probabilistic guarantee, our algorithm needs to issue only $s = \max\{\frac{8p}{\varepsilon^2}\ln\frac{m}{\delta}\ ,\ \frac{12(p-\varepsilon)}{\varepsilon^2}\ln\frac{m}{\delta}\}$[1] queries, which equals $O(\log m)$ when all user parameters are fixed. One feature of s is its independence of n—suggesting that the number of users to probe is irrelevant to the number of users itself. To further extend the usability, we also describe how our algorithm can handle the scenario where users' preferences frequently require updates.

[1] ε is the error parameter and δ is the confidence parameter. They are fully discussed in Section 4.1.

2 Applications

Application 1 (Marketing Survey) It is often a top priority for various industries to conduct marketing surveys to identify their most popular products for promotional purposes. For example, companies in the movie industry are eager to find out the popular movie actors to further promote those actors' movies. The same promotion strategy is also widely employed in various other industries, such as the music and retail industries. Companies in the music industry aim to identify the most popular musical artists to promote their albums, whereas those in the retail industries are keen to discover which brands of a certain type of product (e.g., red wine) are favored by customers.

Although such promotions could bring in substantial profits to the companies, the task of identifying popular products could also be costly due to the huge number of customers needed to probe. For example, the number of unique visitors to the well-known movie website IMDb (`http://www.imdb.com`) exceeds 250 million each month [1]. In addition, companies often have to offer financial incentives (e.g., coupons) to attract participation in a survey. Thus, the costs of conducting surveys include both manpower and funding. As a result, minimizing the number of customers needed to probe should be the goal of a company.

We can model the above marketing survey as our proposed problem. If we aim to discover the popular movie actors, then each customer is modeled as a user $u \in \mathcal{U}$ and each actor as an item $t \in \mathcal{T}$. The aim is therefore to identify all the actors that are favored individually by at least pn customers, while at the same time minimizing the total number of queries required.

Our problem can also discover *sleeping beauties* [11]—items that have potential to become popular but have yet to. For instance, although IMDB already provides a list of popular movies [2], the movies obtained from our problem may not appear in that list, because some movies of popular movie actors may not be popular or well-known. However, such movies possess high potential to go viral because they feature movie actors favored by current users.

Application 2 (Sponsored Search) Another major application of our problem is to help search engine companies uncover popular ads. Given a set of possible search terms—some of which may have never been entered by users before—and a set of ads, search engine companies would want to identify the ads that receive the highest *score* (or *Adrank*, as it is called in the literature) from many search terms. Those ads would therefore be shown frequently to users when they type in keywords appearing in the search term set. To model this application as our proposed problem, we let $\mathcal{U}$ be the set of search terms and $\mathcal{T}$ be the set of ads. The aim is therefore to identify a subset R of ads such that each ad in R receives the highest score from at least pn search terms. At the same time, however, we want to keep the number of score calculations to a minimum. It is worth noting that set R changes over time because any updates of advertisers' bid price for their ads could affect R. To tackle this problem, we will propose a fast update procedure in Section 5.4.

Discovering popular ads can benefit both search engine companies and advertisers. On the companies' side, they may want to raise the price that they charge for popular ads to further increase their revenue. On the advertisers' side, search engine companies can now inform the advertisers concerned that their ads are frequently shown to prospective customers. Given this assurance, those advertisers would more likely keep advertising in the search engines. This information also serves as a valuable indication to the advertisers that their current settings of bid prices and campaign budgets are reasonable. In fact, it is a well-known issue in the sponsored search community that advertisers often struggle to find out whether their current setting is effective in attracting users [6, 24]. Therefore, our problem can alleviate this issue.

Application 3 (Crowdsourcing) Our problem can also help crowdsourcing platforms discover the most popular task types among their users. A crowdsourcing platform allows task

requesters to put tasks of various types (e.g., image tagging and audio transcription) onto the website and allows workers to work on the tasks offered by the requesters [10]. There are several well-established crowdsourcing platforms for both task requesters and workers, such as Amazon Mechanical Turk (AMT) (`http://www.mturk.com/`).

Suppose that a new worker comes to a crowdsourcing platform to look for tasks to work on. However, since the platform does not possess any personal information about the new worker, what type of task can the website recommend to him? A natural step for the website to take is to recommend a set of popular task types that are the favorites of its current workers to the new visitor. In order to identify this set, we can model the scenario as our problem, with the user set $\mathcal{U}$ representing the current workers and the item set $\mathcal{T}$ the task types. Our goal is therefore to discover the task types that are favored by many workers. As with Application 1, because a query requires both manpower and financial incentives, a crowdsourcing company should aim to identify the set of popular task types by probing as few workers as possible.

3 Related Work

3.1 Frequent Item Mining

The *frequent item mining* problem [5, 12, 13] is related to our problem and can be stated as follows: Given a set of items and a set of transactions—each of which is a subset of the set of items—we identify all the itemsets, which are subsets of the item set, such that each of the itemsets appears in at least a given number of the transactions.

For the sake of comparison, we can view the favorite items of a particular user in our problem as a transaction in the frequent item mining problem. There are several crucial differences between the two problems. (1) In our problem, because a user in fact does not make a "transaction" (i.e., the favorite items), it makes sense to assume that the "transaction" is hidden. Therefore, the favorite items of a user remain unknown until we issue a favorite-probing query to identify them. In the frequent item mining problem, by contrast, a transaction is indeed made by a customer and thus it makes sense instead to assume that it is known. Thus, there is no need to issue a favorite-probing query to find the "favorite items" of a user in the frequent item mining problem. In short, while our counterpart assumes that user preferences are known, our problem does not. (2) In our problem, a transaction represents the favorite items of a user. However, in the frequent item mining problem, a transaction generally does not correspond to the favorite items of a user, because, for instance, the items in a transaction might be bought for other people.

3.2 Heavy Hitter Finding

The *heavy hitter finding problem* [7, 8] is similar to the frequent item mining problem, in that a heavy hitter corresponds to a frequent item in frequent item mining. The difference is that in heavy hitter finding, the transactions come as a *data stream*, whereas in traditional frequent item mining, the transactions are fixed and remain constant throughout. The heavy hitter finding problem also assumes that the items in each transaction are known and is therefore different from our problem.

3.3 Survey Sampling

Our problem bears similarities to problems in survey sampling. A particularly relevant one is the *estimation of population proportion* problem [21, 22]. This problem can be stated as follows: given a set of user and a set of categories (or items in the language of our problem), and each user belongs to exactly one category, the aim is to estimate the proportion of users belonging to each category. There are two major differences that distinguish our problem from

the problem in survey sampling. (1) In our problem, a user is allowed to favor multiple items, whereas under the setting of the estimation of population proportion problem, each user can favor *only one* item. Therefore, our problem is far more general than its counterpart. (2) In the estimation of population proportion literature, results are commonly derived based on the following two assumptions [21, 22]: (1) the data is normally distributed, or at least the normal approximation can be applied to the data. (2) The *finite population correction factor* can be ignored. In this paper, we do not make either of these assumptions. Despite the differences above, we compare the performance of the algorithm proposed in [21, 22] with ours in Section 5.1.

4 Problem Definition

Let $\mathcal{U}$ and $\mathcal{T}$ be two given sets, where $\mathcal{U}$ is the *user set* containing $n=|\mathcal{U}|$ users and $\mathcal{T}$ the *item set* containing $m=|\mathcal{T}|$ items. Each user $u\in\mathcal{U}$ is associated with a unique score function that takes as input the item set $\mathcal{T}$ and returns as output the item(s) $t^*\in\mathcal{T}$ that receives the highest score (or highest ranking) from user u, relative to other items in $\mathcal{T}$. We say that a user u favors (or prefers/endorses) the item $t^*\in\mathcal{T}$ if and only if user u gives item t the highest score.

We have at our disposal a *favorite-probing query* q that takes as input a user $u\in\mathcal{U}$ and returns as output the favorite item(s) $q(u)\in\mathcal{T}$ of that user. We measure the *support* of an item $t\in\mathcal{T}$, denoted by support(t), by the number of users who favor that item. We say that an item $t\in\mathcal{T}$ is *frequent* (or sometimes *popular*) if support$(t)\geq pn$, where $p\in(0,1]$ is a parameter called the *support proportion*.

Aim. Given the user set $\mathcal{U}$ and item set $\mathcal{T}$, the HIDDEN FREQUENT ITEM MINING problem is to find a subset $R\subseteq\mathcal{T}$ of the item set such that R contains all the frequent items and no items that are not frequent. Formally, $R=\{t\in\mathcal{T}\,|\,\text{support}(t)\geq pn\}$.

4.1 ε-approximation

We present a relaxed version of the HIDDEN FREQUENT ITEM MINING problem. We call it the ε-approximation of the HIDDEN FREQUENT ITEM MINING problem. We first classify items into different types according to their support. Table 2 provides a summary of the classification.

Definition 1 (Classification of items). *Let $p,\varepsilon\in(0,1]$ be two parameters. We say that an item $t\in\mathcal{T}$ is **frequent** if support$(t)\geq pn$; we say that t is **potentially-frequent** if $(p-\varepsilon)n\leq$ support$(t)<pn$. Otherwise, we say that t is **infrequent**.*

Problem Formulation. An algorithm for the ε-approximation of the HIDDEN FREQUENT ITEM MINING problem accepts three user-specified parameters: (i) support proportion $p\in(0,1]$, (ii) error parameter $\varepsilon\in(0,1]$ such that $\varepsilon<p$, and (iii) failure parameter $\delta\in(0,1]$. Given these parameters, the algorithm provides the following performance guarantee.

Definition 2 (ε-approximation guarantee). *An algorithm is said to achieve the ε-*approximation guarantee *(or guarantee for short) for the* HIDDEN FREQUENT ITEM MINING *problem if, with probability at least $1-\delta$, the algorithm satisfies the following properties simultaneously:*

- ***P1*** *All frequent items (i.e., items whose individual support is at least pn) in item set $\mathcal{T}$ are returned. In other words, the algorithm will produce no **false negatives** (i.e., recall$=100\%$).*
- ***P2*** *No infrequent items (i.e., items whose individual support is less than $(p-\varepsilon)n$) in item set $\mathcal{T}$ are returned. In other words, the algorithm will not return an item that has a support lower than the minimum tolerable value.*

Scenario	Classification (Def. 1)	Decision (Def. 2)
$\text{support}(t) \geq pn$	t is a frequent item	Return t
$(p-\varepsilon)n \leq \text{support}(t) < pn$	t is a potentially-frequent item	May or may not return t (i.e., inconclusive)
$\text{support}(t) < (p-\varepsilon)n$	t is an infrequent item	Do not return t

Table 2. Summary of Definitions 1 and 2.

Remark 1 (Treatment of potentially-frequent items). Notice that potentially-frequent items may or may not be returned. If such an item happens to be returned, then our algorithm is said to have returned a *false positive*—the only scenario in which our algorithm errs. Still, all such false positives possess a desirable property that their support is fairly high: they are only marginally lower than the threshold value sought by the user. In most cases, therefore, returning such a high-support false positive should not be fatal and should be tolerable to the user. As we see in Section 5.3, our proposed algorithm also possesses another desirable property that for every potentially-frequent item, the lower its support, the lower the chance of it being returned.

Example 1. Suppose that $n = 10^5$, $m = 10^3$, $p = 10\%$, $\varepsilon = 1\%$, $\delta = 1\%$. Then, with probability at least $1-\delta = 99\%$, an algorithm that achieves the ε-approximation guarantee returns all items with an individual support at least $pn = 0.1 \cdot 10^5 = 10{,}000$ (Property **P1**). Moreover, the algorithm does not return any items with a support less than $(p-\varepsilon)n = (0.1-0.01)10^5 = 9{,}000$ (Property **P2**). This leaves those items with a support between 9,000 and 10,000; these items may or may not be returned (Remark 1).

5 Algorithm

Notation: discover(t). For every item t, we denote by $\text{discover}(t)$ the support that is *discovered* by our algorithm for item t. This is different from $\text{support}(t)$, which means the *true* support of item t.

5.1 Support-Sampling (SS)

We propose a fast algorithm that provides the ε-approximation guarantee. Our algorithm, termed SUPPORT-SAMPLING (SS), works as follows: It first randomly selects a user, and then it issues a favorite-probing query to identify the user's favorite item(s). It then increments the discovered support of the item(s) accordingly. Our algorithm will repeat the above process for $s = \max\{\frac{8p}{\varepsilon^2}\ln\frac{m}{\delta}, \frac{12(p-\varepsilon)}{\varepsilon^2}\ln\frac{m}{\delta}\}$ times. In other words, it in total selects (without replacement) s users and finds their respective favorite item(s). After sampling all these s users, SS returns all items whose discovered support is at least $(p-\varepsilon/2)s$, along with the probability ($=1-\delta$ if not early terminated) of successfully achieving the ε-approximation guarantee. Notice that the incremental nature of SS makes early termination with a performance guarantee possible. Specifically, at any point of execution, the user can ask SS to return a set of items that (potentially) satisfy the ε-approximation guarantee, together with the probability of success. We call our algorithm *support sampling* because each sample (i.e., a selected user) can be regarded as a piece of supporting evidence that an item is frequent. The pseudocode of SS is given in Algorithm 1.

Comparison with survey sampling Recall from Section 3.3 that our proposed problem shares similarities with the estimation of population proportion problem in survey sampling [21, 22]. In particular, the method proposed in [21, 22] can be adapted to solve a special

Algorithm 1 SUPPORT-SAMPLING (SS)

Input: $\mathcal{U}$,$\mathcal{T}$,p,ε,δ
Output: A set R of items, the probability of success.

$s = \max\left\{\frac{8p}{\varepsilon^2}\ln\frac{m}{\delta}\ ,\ \frac{12(p-\varepsilon)}{\varepsilon^2}\ln\frac{m}{\delta}\right\}$ //Sample size
for each item $t \in \mathcal{T}$ **do**
    discover$(t) = 0$ //Initialization
for $i = 1$ **to** s **do**
    select a user u uniformly at random and independently from $\mathcal{U}$
    issue a favorite-probing query $q(u)$ to identify the favorite item(s) t^* of user u
    increment (all) discover(t^*)
    if user chooses early termination **then**
        //See Lemma 3 for derivation of failure probability
        failure $= \max\left\{m\exp\left(-\frac{i\varepsilon^2}{8p}\right), m\exp\left(-\frac{i\varepsilon^2}{12(p-\varepsilon)}\right)\right\}$
        //R is the set of returned items
        $R = \{t \in \mathcal{T} \mid \text{discover}(t) \geq (p-\varepsilon/2)i\}$
        return (R, 1 − failure) //Early termination
$R = \{t \in \mathcal{T} \mid \text{discover}(t) \geq (p-\varepsilon/2)s\}$
return $(R, 1-\delta)$

case of our problem—when each user favors exactly one item. However, the method becomes *invalid* if any user favors more than one item [21,22]. The method requires a sample size of $\max_{1\leq i\leq m} z^2(1/i)(1-1/i)/\varepsilon^2$, where z is the upper $(\delta/2i)\times 100$th percentile of the standard normal distribution. This sample size is significantly larger than ours when the error tolerance ε is small, which is often the case in practice. For example, our sample size is only half that of our counterpart when $p = 1\%, \varepsilon = 0.1\%, \delta = 1\%, m = 1000$ and for any arbitrary value of n (because both sample sizes are independent of n).

The method in [21,22] has several disadvantages. As detailed in Section 3.3, their results assume that data is normally distributed and the finite population correction factor can be ignored. Our derivation, however, makes no such assumptions. Furthermore, to evaluate their sample size, we need to solve a maximization problem, which is inconvenient in practice. In contrast, our derivation gives rise to a simple closed-form formula that can be readily evaluated.

5.2 Analysis of Support-Sampling

We show that SS provides the ε-approximation guarantee. In the proofs, we allow each user to favor multiple items. Proofs are presented in the Appendix.

Proof strategy. We first consider individually the probability that SS fails to achieve Properties **P1** and **P2**. We then use the *union bound* to find an upper bound for the probability that our algorithm *fails*, whereby deriving the required sample size to achieve the ε-approximation guarantee. We start by proving three lemmas that are useful in showing the ε-approximation guarantee of SS. All proofs can be found in the Appendix.

Lemma 1. SUPPORT-SAMPLING *fails to achieve Property* ***P1*** *of ε-approximation guarantee with probability at most $ke^{-\frac{s\varepsilon^2}{8p}}$, where k is the number of frequent items.*

Lemma 2. SUPPORT-SAMPLING *fails to achieve Property* ***P2*** *of ε-approximation guarantee with probability at most $(m-k)e^{-\frac{s\varepsilon^2}{12(p-\varepsilon)}}$.*

Lemma 3. SUPPORT-SAMPLING *fails to achieve the ε-approximation guarantee with probability at most $\max\{me^{-\frac{s\varepsilon^2}{8p}}, me^{-\frac{s\varepsilon^2}{12(p-\varepsilon)}}\}$.*

Theorem 1. SUPPORT-SAMPLING *achieves the ε-approximation guarantee by sampling s users, where $s = \max\{\frac{8p}{\varepsilon^2}\ln\frac{m}{\delta}\ ,\ \frac{12(p-\varepsilon)}{\varepsilon^2}\ln\frac{m}{\delta}\}$.*

Proof. See Appendix.

Remark 2. Theorem 1 implies that if all parameters are fixed, the sample size s becomes $O(\log m)$.

Example 2 (Performance Comparison). Table 3 shows the number of queries needed by SS under a variety of input settings. SS consistently requires only a tiny number of queries—typically less than 0.5% of that needed by the naive one.

Input					**Sample Size**	**Ratio**
p	ε	δ	n	m	s	s/n
.1	.02	.05	10^7	10^7	$4.6{\cdot}10^4$	0.46%
.2	**.04**	.05	10^7	10^7	$2.3{\cdot}10^4$	0.23%
.2	**.06**	.05	10^7	10^7	$0.9{\cdot}10^4$	0.09%
.1	.02	**.01**	10^7	10^7	$5.0{\cdot}10^4$	0.50%
.1	.02	.05	$\mathbf{10^8}$	10^7	$4.6{\cdot}10^4$	0.05%
.1	.02	.05	$\mathbf{10^9}$	10^7	$4.6{\cdot}10^4$	0.005%
.1	.02	.05	10^7	$\mathbf{10^8}$	$5.1{\cdot}10^4$	0.51%
.1	.02	.05	10^7	$\mathbf{10^9}$	$5.7{\cdot}10^4$	0.57%

Table 3. Sample size needed by SS under various input settings. Input in the first row is the default; numbers in bold are values different from the default.

5.3 Analysis of potentially-frequent items

In this section, we give further insight into how often SS returns potentially-frequent items—those with a support marginally lower than pn. Our result shows that if a potentially-frequent item t has a support between $(p-\varepsilon/2)n$ and pn, then it is *likely* that it will be returned. On the contrary, if its support is between $(p-\varepsilon)n$ and $(p-\varepsilon/2)n$, then it is *unlikely* that it will be returned. More generally, our results signify that for every potentially-frequent item, the lower its support, the less likely our algorithm will return it. Therefore, SS possesses another desirable property that if a potentially-frequent item happens to be returned, then the chance is higher that it has a support close to pn rather than to $(p-\varepsilon)n$. Our discussion is supported by the results below. Proofs are presented in the Appendix.

Lemma 4. *Let random variables X and Y be the discovered support of items t_h and t_ℓ respectively, such that $X \sim B(s,p_h)$ and $Y \sim B(s,p_\ell)$, where $p_h = \text{support}(t_h)/n$ and $p_\ell = \text{support}(t_\ell)/n$. If $\text{support}(t_\ell) \le \text{support}(t_h)$, or equivalently $p_\ell \le p_h$, then*

$$\boldsymbol{Pr}[Y \ge c] \le \boldsymbol{Pr}[X \ge c], \text{ where } c = (p-\varepsilon/2)s.$$

Lemma 4 suggests that as the support of a potentially-frequent item decreases, so does the chance of SS returning it.

We now derive tight bounds for the probability that a potentially-frequent item is returned.

Proposition 1. *Let the support of a given potentially-frequent item t be $pn - r$, where $\varepsilon n/2 < r \le \varepsilon n$. The probability that item t is returned by SS is* ***at most*** $e^{-2s\left(\frac{r}{n}-\frac{\varepsilon}{2}\right)^2}$.

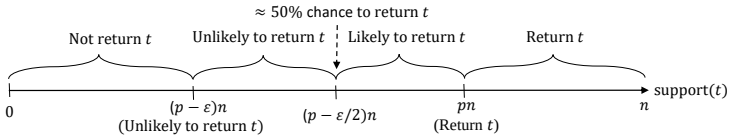


Fig. 1. Decisions by SUPPORT-SAMPLING for any item t.

Example 3. It is easy to see that the bound decreases exponentially as r increases from $\varepsilon n/2$ to εn. To examine how rapidly the bound falls, we adopt the default input in Table 3. Now, if $r=\frac{\varepsilon n}{1.5}$, then the probability that an item with a support $=pn-\frac{\varepsilon n}{1.5}=(p-\frac{\varepsilon}{1.5})n$ is returned is at most 1.6×10^{-3}. As r increases from $\frac{\varepsilon n}{1.5}$ to εn, the probability decreases exponentially from 1.6×10^{-3} to 6.3×10^{-26}.

Our next result shows that if an item has support between $(p-\varepsilon/2)n$ and pn, it is *likely* to be returned.

Proposition 2. *Let the support of a given potentially-frequent item t be $pn-r$, where $0<r<\varepsilon n/2$. The probability that item t is returned by SS is* ***at least*** $1-e^{-2s\left(\frac{r}{n}-\frac{\varepsilon}{2}\right)^2}$.

Special Case: $r=\varepsilon n/2$. A glance at Propositions 1 and 2 suggests that both are *undefined* for $r=\varepsilon n/2$. The reason is that for an item t with support$(t)=(p-\varepsilon/2)n$, its *expected* discovered support equals $(p-\varepsilon/2)s$, and our algorithm returns item t only if its discovered support is at least $(p-\varepsilon/2)s$—which is exactly the expected discovered support. Unfortunately, in general, a tail inequality—including Hoeffding's inequalities—is only capable of bounding the probability that the value assumed by a random variable *exceeds* or *falls behind* the expected value. Thus, we cannot use a tail inequality to bound the probability that item t is returned.

Even so, we argue that the probability of returning item t is approximately 0.5 by appealing to the *Central Limit Theorem* [23]. Notice that the discovered support of item t follows the binomial distribution $B(s,p-\varepsilon/2)$. Since s is typically larger than 30, which is a rule of thumb for applying the Center Limit Theorem [23], the Central Limit Theorem tells us that the binomial distribution is approximately a normal distribution with mean $\mu=(p-\varepsilon/2)s$ [23]. Now, because the probability for a normally distributed random variable to be at least as large as its mean is 0.5, it follows that the probability of item t being returned is approximately 0.5. Figure 1 summarizes our discussion in this section.

5.4 Updates of users' preferences

In some applications, a user's preference can be ever-changing. In the sponsored search application, for example, the score that a search term assigns to an ad will change accordingly whenever the advertiser concerned adjusts the bid price for that ad. This therefore motivates us to devise a fast update method. Fortunately, SUPPORT-SAMPLING is robust to updates of users' preferences. In fact, not only can we reuse the results based on the previously sampled users, we also need not sample any *new* users to fulfill the ε-approximation guarantee. Specifically, for each user u whose preference needs to be updated, we consider two cases.

Case 1: u was sampled. We issue an additional query to identify the new favorite item(s) t of user u and increment discovered(t). we also decrement the discovered support of the item(s) that was u's favorite.

Case 2: u was not sampled. We do not need to issue any additional query, nor do we need to update the result of our algorithm.

Input	Values
n	1m, 2m, 4m, **8m**, 10m
m	40k, 80k, 160k, **320k**, 640k
p	0.05,**0.10**,0.15,0.20,0.25,0.30
ε	$\mathbf{0.1}\boldsymbol{p}$,$0.15p$,$0.2p$,$0.25p$,$0.3p$

Table 4. Input values (defaults shown in bold).

Correctness. After the above procedure, SS still maintains the ε-approximation guarantee. For Case 1, the score function update of a sampled user u will not affect the selection probability of any user, because the selection probability depends only on the number of users, rather than on the users' score functions. For Case 2, we note that user u's preference will not be considered with or without a score function update, because (1) u was not sampled and (2) the update will not affect the selection probability, as in Case 1.

Example 4. Suppose we update the preferences of k ($1 \leq k \leq n$) users, each of whom is randomly chosen without replacement. While the naive one needs to issue exactly k queries, our algorithm needs to issue a query only if the user was sampled previously. So, the number of queries needed by our algorithm for the update follows a *hypergeometric* distribution [23]. Thus, the expected number of queries needed is $k \times (s/n)$. Using the default setting shown in Table 3, we have $s/n \approx 0.46\%$. Therefore, the expected number of queries is just 0.46% of the number required by the naive algorithm.

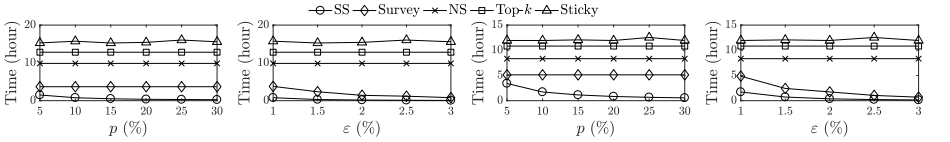
6 Experiments

We experimentally evaluate our proposed algorithms using both real and synthetic datasets.

Setup. All experiments were run on a machine with a 3.4GHz CPU and 32 GB memory. The OS is Linux (CentOS 6). All algorithms were coded in C++ and compiled using g++.

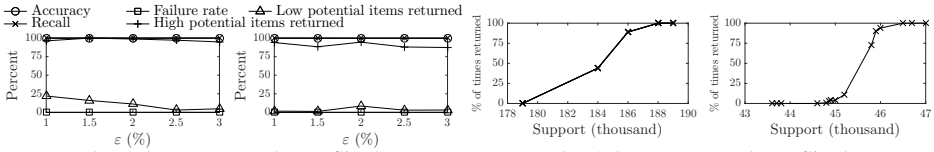
We compare our proposed algorithms, SUPPORT-SAMPLING (SS), with the Naive Solution (NS) as well as three adapted existing algorithms: Survey Sampling [21, 22] (Section 3.3), Top-k query [15, 19] (a well-known problem in the database literature) and Sticky Sampling [17], which is a classic algorithm for the heavy hitter problem (Section 3.2). The adaption of Top-k query and Sticky Sampling is made possible by requiring the algorithms to issue a favorite-probing query for each user to identify the user's favorite items as a first step. In particular, because Sticky Sampling is a streaming algorithm, we implement it in such a way that a new user arrives only after we finish processing the user immediately preceding him/her, so as to allow enough time for it to discover the favorite items of each user. The performance metrics are the execution time and accuracy. For randomized algorithms such as SS and Sticky Sampling, we ran them 100 times for each experiment setting to report its average execution time and average accuracy. The default values of the parameters p and ε are 10% and 1% respectively.

Real Datasets. We use two real datasets to simulate the marketing survey application (Application 1): Yahoo! musical artist ratings dataset [4] (Yahoo! dataset) and Netflix movie ratings dataset [3] (Netflix dataset). Yahoo! dataset contains over 100 million user ratings of 98,211 musical artists ($m = 98{,}211$) by 1,948,882 users ($n = 1{,}948{,}882$), whereas Netflix dataset consists of more than 100 million users ratings of 17,770 movies ($m = 17{,}770$) by 480,189 users ($n = 480{,}189$). In real datasets, a query $q(u)$ on user u corresponds to finding the artists (or movies for the Netflix dataset) that receive the highest rating from user u, relative to other artists/movies. We assume those artists/movies that did not receive a rating from user u were not favored by u. We must also emphasize that although the ratings given by each user are *known* before we issue a query, in practice this information is *unknown* beforehand. To account for the need to obtain user ratings in practice, we multiply the execution time of each algorithm by $10{,}000$.



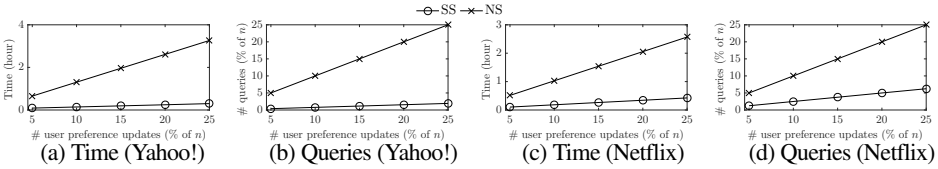
(a) Varying p (Yahoo!) (b) Varying ε (Yahoo!) (c) Varying p (Netflix) (d) Varying ε (Netflix)

Fig. 2. Performance of SS versus other competitors on real datasets.



(a) Yahoo! dataset (b) Netflix dataset (c) Yahoo! dataset (d) Netflix dataset

Fig. 3. Statistics of SS on real datasets.



(a) Time (Yahoo!) (b) Queries (Yahoo!) (c) Time (Netflix) (d) Queries (Netflix)

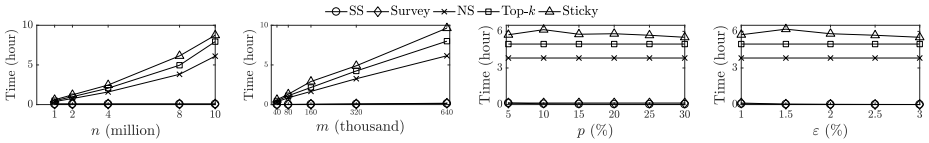
Fig. 4. SS vs. NS on updating users' preferences on real datasets.

Synthetic Datasets. We use synthetic datasets to simulate the sponsor search application (Application 2). Following related studies (e.g., [18, 25]), we view each item $t \in \mathcal{T}$ as a point in a space, and we set the dimension (i.e., number of attributes) d to 5. To exploit the generality of our problem, each user is randomly associated with either a linear $\sum_{i=1}^{d} w_i x_i$ or quadratic $\sum_{i=1}^{d} w_i x_i^2$ score function, where w_i is the attribute weight of the score function for the ith attribute of an item, and x_i is the ith attribute value of an item. Note that although linear score functions are often employed in the literature (e.g., [18, 25]) for its simplicity, non-linear score functions, such as quadratic ones used in our experiments, can better model users' preferences in many cases [16]. We independently generate each attribute value of an item from the uniform distribution with support $(0,1)$, and so is each attribute weight w_i of a user's score function.

In synthetic datasets, a query $q(u)$ on user u corresponds to computing $\text{argmax}_{t \in \mathcal{T}} \text{score}(u,t)$, where $\text{score}(u,t)$ is the score that user u gives to item t. Experiments on synthetic datasets were conducted using various input settings shown in Table 4. The settings are similar to existing studies (e.g., [18, 25]), except that dataset sizes are proportionally increased. As in [25], we set m (number of items) to be smaller than n (number of users). This is because in practice—and also in our real datasets—the item set size is often substantially smaller than the user set size. In all experiments, the failure parameter δ is set to 5%, due to its negligible impact on the execution time.

6.1 Results on Real Datasets

We compare the execution time of various algorithms and then discuss the accuracy of SS on real datasets.



(a) Time as n varies (b) Time as m varies (c) Time as p varies (d) Time as ε varies

Fig. 5. Performance of SS versus other competitors on synthetic datasets.

Comparison on efficiency. As Figure 2 shows, SS consistently requires far less time than NS and other competitors, and their performance gap widens as p or ε increases. In particular, when both p and ε are at their default values, SS takes only about 8% of the time by NS to process the Yahoo! dataset, and about 20% of the time by the survey method, which is the second best. Notice that Top-k and Sticky Sampling are even more inefficient than NS because in order to adopt them, it is necessary to first issue a favorite-probing query for each user to identify the user's favorite items, which is exactly the strategy used by NS.

Accuracy of SS. We start by introducing some terms. We call a potentially-frequent item t a *high potential item* if $(p-\varepsilon/2)n \leq \text{support}(t) < pn$; otherwise, we call it a *low potential item*.

Figure 3 shows the statistics of SS on real datasets. The *Failure rate* in Figures 3(a) and (b) refers to the proportion of times SS fails to achieve the ε-approximation guarantee. Observe that although the failure parameter δ is set to 5%, SS never failed to achieve the ε-approximation guarantee in all the experiments, and thus recall is always 100%. This is because our theoretical analysis for SS is conservative, and so *in practice* the failure probability is generally much lower than the failure parameter δ. The accuracy of SS is consistently higher than 99.7% in all experiments. The very slight inaccuracy stems from the fact that some potentially-frequent items are returned. However, notice that those returned potentially-frequent items are often high potential items, rather than low potential ones. This also verifies our findings in Section 5.3: high potential items are likely to be returned, whereas low potential items are unlikely to. Figures 3(c) and (d) show the fraction of times over the 100 runs that each potentially-frequent item is returned, when parameters p and ε are at their default values.

Comparison on updating users' preferences We compare the performance of SS and NS on updating users' preferences. Recall from Section 5.4 that SS can handle updates of users' preferences efficiently, whereas NS needs to issue an additional query for each user whose preference requires an update. Figures 4(a) and (b) show that for the Yahoo! dataset, both the execution time and number of queries needed by NS for updating users' preferences grow far more rapidly than SS. Similar trends can also be observed from the Netflix dataset (Figures 4(c) and (d)).

6.2 Results on Synthetic Datasets

We compare SS with other algorithms on synthetic datasets with sizes specified in Table 4.

Scalability with n. The execution time of NS, Top-k query and Sticky Sampling grow linearly with n (Figures 5(a)). The linear relationship makes them impractically slow when n is large. For instance, NS requires more than 6 hours to complete when n reaches 10M. In stark contrast to its competitors, SS is *insensitive* to the increase of n, consistently taking less than 4 minutes to execute regardless of the size of n.

Scalability with m. Figure 5(b) shows that the execution time of our competitors grows linearly with m. For example, NS takes almost 6.2 hours to process 640K items. By contrast,

the execution time taken by SS grow extremely slowly with m, and it requires less than 8 minutes to process 640K items. The efficiency of SS can be explained by referring to the sample size formula $s = \max\{\frac{8p}{\varepsilon^2}\ln\frac{m}{\delta}, \frac{12(p-\varepsilon)}{\varepsilon^2}\ln\frac{m}{\delta}\}$, where s grows only logarithmically with m.

Influence of support proportion p. As Figures 5(c) show, SS requires less time as p increases. The cost of our competitors remain constant because of their independence of p.

Influence of error rate ε. Similar to the situation when p increases, the cost of SS falls as ε increases (Figures 5(d)). In particular, SS requires only about 1.5% of the execution time and queries by NS when $\varepsilon = 1\%$.

7 Conclusion

We proposed an interesting problem called HIDDEN FREQUENT ITEM MINING, which has important applications in various fields.We devise a sampling algorithm that sacrifices slight accuracy in exchange for substantial improvement in efficiency.

Acknowledgement. The research is supported by HKRGC GRF 14205117.

References

1. About IMDb. http://www.imdb.com/pressroom/about/, 2019.
2. IMDb Charts. http://www.imdb.com/chart/top, 2019.
3. Netflix datasets. www.netflixprize.com, 2019.
4. Yahoo! datasets. https://webscope.sandbox.yahoo.com/, 2019.
5. R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.
6. D. Chakrabarty, Y. Zhou, and R. Lukose. Budget constrained bidding in keyword auctions and online knapsack problems. In *WWW2007, Workshop on Sponsored Search Auctions*, 2007.
7. G. Cormode and M. Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2):1530–1541, 2008.
8. X. Dimitropoulos, P. Hurley, and A. Kind. Probabilistic lossy counting: an efficient algorithm for finding heavy hitters. *ACM SIGCOMM Computer Communication Review*, 38(1):5–5, 2008.
9. D. P. Dubhashi and A. Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.
10. M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 61–72. ACM, 2011.
11. E. Garfield. Premature discovery or delayed recognition-why? *Current Contents*, (21):5–10, 1980.
12. J. Han, J. Pei, and M. Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.
13. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *ACM Sigmod Record*, volume 29, pages 1–12. ACM, 2000.
14. W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.
15. I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*, 40(4):11, 2008.
16. T. Kessler Faulkner, W. Brackenbury, and A. Lall. k-regret queries with nonlinear utilities. *Proceedings of the VLDB Endowment*, 8(13):2098–2109, 2015.
17. G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 346–357. VLDB Endowment, 2002.
18. P. Peng and R. C.-W. Wong. k-hit query: Top-k query with probabilistic utility function. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015.
19. A. Shanbhag, H. Pirk, and S. Madden. Efficient top-k query processing on massively parallel hardware. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1557–1570. ACM, 2018.

20. S. K. Solanki and J. T. Patel. A survey on association rule mining. In *2015 Fifth International Conference on Advanced Computing & Communication Technologies*, pages 212–216. IEEE, 2015.
21. S. K. Thompson. Sample size for estimating multinomial proportions. *The American Statistician*, 41(1):42–46, 1987.
22. S. K. Thompson. *Sampling*. Wiley, 2012.
23. R. E. Walpole, R. H. Myers, S. L. Myers, and K. Ye. *Probability and statistics for engineers and scientists*. Macmillan New York, 2011.
24. W. Zhang, Y. Zhang, B. Gao, Y. Yu, X. Yuan, and T.-Y. Liu. Joint optimization of bid and budget allocation in sponsored search. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1177–1185. ACM, 2012.
25. Z. Zhang, C. Jin, and Q. Kang. Reverse k-ranks query. *Proceedings of the VLDB Endowment*, 2014.

A Appendix: Probabilistic Inequalities

We outline several probabilistic inequalities that will be employed in our proofs.

Proposition 3 (Chernoff's bound [9] and Hoeffding's inequality [14]). *Let $X_1,X_2,...,X_s$ be independent Bernoulli variables and $\boldsymbol{Pr}[X_i=1]=p$, where $0<p<1$, for $i=1,2,...,s$. Let $X=\sum_{i=1}^{s}X_i$ and $\boldsymbol{E}[X]=\mu$ such that $\mu_\ell\leq\mu\leq\mu_h$, where $\mu_\ell,\mu_h\in\mathbb{R}$. Then, for any $\varepsilon<1$,*

$$\boldsymbol{Pr}[X\geq(1+\varepsilon)\mu_h]\leq e^{\frac{-\mu_h\varepsilon^2}{3}},\quad \boldsymbol{Pr}[X\leq(1-\varepsilon)\mu_\ell]\leq e^{\frac{-\mu_\ell\varepsilon^2}{2}}, \tag{1}$$

$$\boldsymbol{Pr}[X\geq\mu+\varepsilon]\leq e^{-\frac{2\varepsilon^2}{s}},\quad \boldsymbol{Pr}[X\leq\mu-\varepsilon]\leq e^{-\frac{2\varepsilon^2}{s}}. \tag{2}$$

B Appendix: Proofs

Proof (Lemma 1). Let X be the discovered support of a given frequent item t (i.e., an item whose support is at least pn). For $i=1,2,...,s$, let X_i be the indicator random variable such that $X_i=1$ if the ith sampled user favors item t, and $X_i=0$ otherwise. We first note that $X=\sum_{i=1}^{s}X_i$. In addition, since our algorithm samples users independently and uniformly at random, we have $\mathbf{E}[X_i]=\mathbf{Pr}[X_i=1]\geq\frac{pn}{n}=p$, for all $i=1,2,...,s$. So, we conclude that $\mathbf{E}[X]=\mathbf{E}[\sum_{i=1}^{s}X_i]=\sum_{i=1}^{s}\mathbf{E}[X_i]\geq ps$.

Now, we bound the probability that item t is not returned. According to our algorithm, this event occurs if and only if the discovered support of item t is less than $(p-\varepsilon/2)s$:

$$\mathbf{Pr}\left[X<\left(p-\frac{\varepsilon}{2}\right)s\right]=\mathbf{Pr}\left[X<\left(1-\frac{\varepsilon}{2p}\right)ps\right]\leq e^{-\frac{ps\left(\frac{\varepsilon}{2p}\right)^2}{2}}=e^{-\frac{s\varepsilon^2}{8p}}.$$

The inequality above is established by Chernoff's bound (Proposition 3). Now, without loss of generality, suppose that there are k frequent items in the item set $\mathcal{T}$, where $1\leq k\leq m$ is unknown. As will be clear shortly, the value of k is irrelevant to the analysis of our algorithm. By using the union bound (a.k.a Boole's inequality), therefore, the probability that at least one of the k frequent items is not returned by our algorithm is bounded above by $ke^{-\frac{s\varepsilon^2}{8p}}$. □

Proof (Lemma 2). Now, let Y be the discovered support of a given infrequent item (i.e., an item whose support is less than $(p-\varepsilon)n$). By using the same line of argument for random variable X in Lemma 1, we can show that $\mathbf{E}[Y]<(p-\varepsilon)s$.

Now, we bound the probability that the given infrequent item is returned. By the design of our algorithm, this event occurs if and only if the discovered support of that item is at least $(p-\varepsilon/2)s$:

$$\mathbf{Pr}\left[Y\geq\left(p-\frac{\varepsilon}{2}\right)s\right]=\mathbf{Pr}\left[Y\geq\left(1+\frac{\varepsilon}{2(p-\varepsilon)}\right)(p-\varepsilon)s\right]\leq e^{-\frac{(p-\varepsilon)s\left[\frac{\varepsilon}{2(p-\varepsilon)}\right]^2}{3}}=e^{-\frac{s\varepsilon^2}{12(p-\varepsilon)}}.$$

The inequality is established by Chernoff's bound (Proposition 3). Since there are k items whose support is at least pn, it follows that there are at most $m-k$ items whose support is less than $(p-\varepsilon)n$ (because there are m items). By the union bound, the probability that our algorithm returns at least one of the infrequent items is at most $(m-k)e^{-\frac{s\varepsilon^2}{12(p-\varepsilon)}}$. □

Proof (Lemma 3). From Lemma 1 and Lemma 2, we know that the probability that our algorithm fails to achieve Property **P1** (resp. Property **P2**) of the ε-approximation guarantee is at most $ke^{-\frac{s\varepsilon^2}{8p}}$ (resp. $(m-k)e^{-\frac{s\varepsilon^2}{12(p-\varepsilon)}}$). By appealing to the union bound again, the probability that our algorithm fails to achieve the ε-approximation guarantee is at most $ke^{-\frac{s\varepsilon^2}{8p}}+(m-k)e^{-\frac{s\varepsilon^2}{12(p-\varepsilon)}}$. We simplify the expression to identify the sample size s:

$$e^{-\frac{s\varepsilon^2}{8p}} \leq e^{-\frac{s\varepsilon^2}{12(p-\varepsilon)}} \iff 12(p-\varepsilon) \geq 8p \iff p \geq 3\varepsilon \,. \tag{3}$$

Case 1: $p<3\varepsilon$. By referring to Equations (3), we know that $e^{-\frac{s\varepsilon^2}{8p}} > e^{-\frac{s\varepsilon^2}{12(p-\varepsilon)}}$. Hence

$$ke^{-\frac{s\varepsilon^2}{8p}}+(m-k)e^{-\frac{s\varepsilon^2}{12(p-\varepsilon)}} < ke^{-\frac{s\varepsilon^2}{8p}}+(m-k)e^{-\frac{s\varepsilon^2}{8p}} = me^{-\frac{s\varepsilon^2}{8p}}.$$

Case 2: $p\geq 3\varepsilon$. Again by Equations (3), we have

$$ke^{-\frac{s\varepsilon^2}{8p}}+(m-k)e^{-\frac{s\varepsilon^2}{12(p-\varepsilon)}} \leq ke^{-\frac{s\varepsilon^2}{12(p-\varepsilon)}}+(m-k)e^{-\frac{s\varepsilon^2}{12(p-\varepsilon)}} = me^{-\frac{s\varepsilon^2}{12(p-\varepsilon)}}.$$

Taking the maximum of both cases, we finish the proof. □

Proof (Theorem 1). We use the result of Lemma 3.

Case 1: $p<3\varepsilon$. In this case, the probability that SUPPORT-SAMPLING fails is at most $me^{-\frac{s\varepsilon^2}{8p}}$. By setting this quantity to δ and then solve for s, we have $s=\frac{8p}{\varepsilon^2}\ln\frac{m}{\delta}$.

Case 2: $p\geq 3\varepsilon$. Similarly, the probability that SUPPORT-SAMPLING fails in this case is at most $me^{-\frac{s\varepsilon^2}{12(p-\varepsilon)}}$. By setting this quantity to δ and then solve for s, we have $s=\frac{12(p-\varepsilon)}{\varepsilon^2}\ln\frac{m}{\delta}$. Taking the maximum of both cases, we finish the proof. □

Proof (Lemma 4). For $i=1,2,...,s$, let X_i and Z_i be Bernouilli distributions with mean p_h and p_ℓ/p_h, respectively. Let $Y_i := X_iZ_i$ so that $Y_i \leq X_i$ with probability 1, and that Y_i is a Bernouilli distribution with mean p_ℓ. Now, notice that $X=\sum_{i=1}^n X_i$ and $Y=\sum_{i=1}^n Y_i$. Also, since $Y_i \leq X_i$, it follows that $Y \leq X$ with probability 1. Therefore, the event $Y \geq c$ implies the event $X \geq c$. Consequently, we have $\mathbf{Pr}[Y \geq c] \leq \mathbf{Pr}[X \geq c]$. □

Proof (Proposition 1). Let Y be the discovered support of a given potentially-frequent item t with support$(t)=pn-r$, where $\varepsilon n/2 < r \leq \varepsilon n$. Then, using the same argument in Lemma 1, we can show that $\mathbf{E}[Y]=\left(\frac{pn-r}{n}\right)s=\left(p-\frac{r}{n}\right)s$. Hence, the probability that item t is returned by SUPPORT-SAMPLING is given by $\mathbf{Pr}\left[Y \geq \left(p-\frac{\varepsilon}{2}\right)s\right] = \mathbf{Pr}\left[Y \geq \left(p-\frac{\varepsilon}{2}+\frac{r}{n}-\frac{r}{n}\right)s\right] = \mathbf{Pr}\left[Y \geq \left(p-\frac{r}{n}\right)s+\left(\frac{r}{n}-\frac{\varepsilon}{2}\right)s\right] = \mathbf{Pr}\left[Y \geq \mathbf{E}[Y]+\left(\frac{r}{n}-\frac{\varepsilon}{2}\right)s\right]$. Now, since $r > \frac{\varepsilon n}{2}$, it follows that $\left(\frac{r}{n}-\frac{\varepsilon}{2}\right)s > 0$. Hence, we can apply Hoeffding's inequality (Proposition 3) to bound the equation: $\mathbf{Pr}\left[Y \geq \mathbf{E}[Y]+\left(\frac{r}{n}-\frac{\varepsilon}{2}\right)s\right] \leq e^{-\frac{2\left(\frac{r}{n}-\frac{\varepsilon}{2}\right)^2 s^2}{s}} = e^{-2s\left(\frac{r}{n}-\frac{\varepsilon}{2}\right)^2}$. □

Proof (Proposition 2). The proof is similar to that of Proposition 1 and is omitted due to the space constraint. □