# Fast, Accurate, and Space-efficient Tracking of Time-weighted Frequent Items from Data Streams

Yongsub Lim
KAIST
yongsub@kaist.ac.kr

Jihoon Choi
KT
jihoon.choi@kt.com

U Kang
KAIST
ukang@cs.kaist.ac.kr

## ABSTRACT

How can we discover interesting patterns from time-evolving high speed data streams? How to analyze the data streams quickly and accurately, with little space overhead? High speed data stream has been receiving increasing attentions due to its wide applications such as sensors, network traffic, social networks, etc. One of the most fundamental tasks in the data stream is to find frequent items; especially, finding recently frequent items has become important in real world applications.

In this paper, we propose TwMinSwap, a fast, accurate, and space-efficient method for tracking recent frequent items. TwMinSwap is a deterministic version of our motivating algorithm TwSample which is a sampling based randomized algorithm with nice theoretical guarantees. TwMinSwap improves TwSample in terms of speed, accuracy, and memory usage. Both require only $O(k)$ memory spaces, and do not require any prior knowledge on the stream such as its length and the number of distinct items in the stream. Through extensive experiments, we demonstrate that TwMinSwap outperforms all competitors in terms of accuracy and memory usage, with fast running time. Thanks to TwMinSwap, we report interesting discoveries in real world data streams, including the difference of trends between the winner and the loser of U.S. presidential candidates, and doubly-active patterns of movies.

## Categories and Subject Descriptors

H.2.8 [**Information Systems**]: Database Applications—*Data mining*

## General Terms

Design, Experimentation, Algorithms

## Keywords

Data Stream, Time-weighted Counting, Sampling, Frequent Items, Hot Items, Top-$k$ Items


*CIKM'14,* November 3–7, 2014, Shanghai, China.

http://dx.doi.org/10.1145/2661829.2662006.

## 1. INTRODUCTION

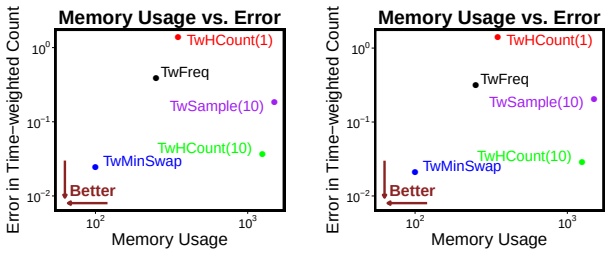
How can we discover currently hot items in high speed data streams, like keyword streams from social networks or restaurant check-ins? How to track them in real time with high accuracy and small memory requirements? These questions are directly related to finding recently frequent items in a data stream. Formally, a data stream is defined by a sequence of items where each item arrives one by one. Usually, the length of the stream and the number of distinct items are considered as very large numbers, possibly infinite. Due to this massiveness, it is impossible to store all the information from the stream, and thus it becomes important to efficiently use memory spaces. As a result, most data stream mining algorithms perform approximation rather than exact computation, and the followings are generally required [3]. First, the stream should be scanned as few times as possible: only one scan is enough for many recent algorithms. Second, the amount of used memory spaces should be limited and independent of the number of distinct items and the stream length, e.g. $O(k)$ space complexity for finding top-$k$ frequent items. Third, processing an item at each time should be fast because the rate of item arrival can be bursty, e.g. Internet traffic may be exploded by network anomalies like DDoS (Distributed Denial of Service) attacks. Fourth, an up-to-date result should be available on demand. These requirements allow that data stream mining algorithms run in real time with small memory spaces.

A number of studies [4, 5, 11, 19] have shown efficacy of their methods in finding frequent items in a data stream, but still it is not clear whether they can find *recent* frequent items correctly. Although an item whose frequency decreases over time tends to have a small count by construction of algorithms, it is not done explicitly. Another problem is that among discovered frequent items, it is hard to know when they have become frequent. Depending on applications, the problem is crucial. For example, when we monitor keywords mentioned in SNS, it is important to know which one is the current trend and which one is the past trend. To overcome this weakness, finding recent frequent items from a data stream has been also studied [1, 8, 12]. However, they have limitations in accuracy, running time, and memory usage.

In this paper, we propose TwMinSwap, a fast, accurate, and space-efficient method for tracking recent frequent items. TwMinSwap is a deterministic version of our motivating algorithm TwSample which is a sampling based randomized algorithm with nice theoretical guarantees. TwMinSwap improves TwSample in terms of speed, accuracy, and

**Table 1: Comparison of performance of our algorithms and competitors. For each row, we write the best in bold and the worst with the canceled line. Our proposed deterministic algorithm TwMinSwap outperforms the others in precision & recall, error in estimation of time-weighted counts, and memory usage. In running time, it is the second best one.**

| | [Recommended] | | Competitors | |
|---|---|---|---|---|
| | TwMinSwap (Sec. 3.2) | TwSample (Sec. 3.1) | TwFreq [25] | TwHCount [4] |
| Precision & Recall | **Highest** | Low | ~~Lowest~~ | High |
| Error in TwCount | **Lowest** | Medium | ~~Highest~~ | **Lowest** |
| Memory Usage | **Smallest** | ~~Largest~~ | Small | ~~Largest~~ |
| Time | Fast | ~~Slowest~~ | Medium | **Fastest** |




(a) Static Distribution (b) Dynamic Distribution

**Figure 1: Our proposed deterministic algorithm TwMinSwap outperforms the others in both memory usage and error of estimated time-weighted counts of top-$k$ items. Despite comparable estimation of TwHCount(10) to that of TwMinSwap, TwHCount(10) requires much more memory spaces.**

memory usage. Both algorithms only require $O(k)$ space complexity. Especially, our deterministic algorithm TwMinSwap requires no other parameter than $k$ and $\alpha$, and this simplicity enables to not only save memory spaces but also reduce per-item processing time. Table 1 compares our proposed TwMinSwap with other competitors, and Figure 1 shows the plots of memory usage vs. error in estimated time-weighted counts for them. TwMinSwap outperforms the others in terms of precision & recall, time-weighted count estimation, and memory usage; its speed is comparable to the fastest competitor TwHCount requiring large memory spaces (see also Figure 4).

Conducting extensive experiments to compare with existing methods, we demonstrate that our algorithms find top-$k$ time-weighted frequent items with small memory spaces whose error is small in terms of precision & recall, and estimated time-weighted counts. Also applying the algorithms to real world data streams, we show that tracking recently frequent activities and keywords enables to discover sudden bursts of attentions to currently hot events and trends in real time.

Our contributions are summarized as follows.

1. **Method.** We introduce time-weighted counting and propose TwMinSwap, an algorithm to find top-$k$ time-weighted frequent items from a data stream which is a deterministic variation of our motivating randomized algorithm TwSample.
2. **Performance.** We show that (1) TwMinSwap outperforms in precision & accuracy and time-weighted count estimation, and that (2) the speed of TwMinSwap is comparable to that of the fastest competitor TwHCount. Especially, TwMinSwap is 1.5× better in time-weighted count estimation and 2.5× better in memory usage than the second best competitors for the two metrics.
3. **Discovery.** We apply TwMinSwap to real world data streams and show interesting discoveries from real world data streams: they include difference of trends between the winner and the loser of U.S. presidential candidates, and doubly-active patterns of movies when they are released at theaters and in DVDs.

The rest of the paper is organized as follows. In Section 2, we give related works including algorithms with and without time-weighting. In Section 3, we describe and analyze our proposed method. In Section 4, experimental comparison with other competitors on synthetic data streams is provided, and in Section 5, we show the discovery results of applying our method to real world data streams. Finally we conclude our work in Section 6.

Table 2 lists the symbols frequently used in this paper.

**Table 2: Table of symbols.**

| Symbol | Description |
|---|---|
| $N$ | stream length |
| $n$ | # of distinct items |
| $k$ | # of (time-weighted) frequent items |
| $\alpha$ | time decaying factor |
| $u, v$ | item identifiers |
| $c, c_u$ | counter (of $u$) |
| $T_u$ | set of timestamps that $u$ occurs |
| $W(u)$ | time-weighted count of $u$ |
| $P(u)$ | penalized time-weighted count of $u$ |
| $t, t_i$ / $t_{cur}$ | timestamps / current timestamp |
| $K$ | set of discovered (time-weighted) frequent items |
| $\lambda$ | penalty term for $P(u)$ |
| $\sigma$ | increase ratio of $\lambda$ |

# 2. RELATED WORKS

In this section, we describe related works on finding frequent items, and *recent* frequent items.

## 2.1 Finding Frequent Items

There have been numerous studies to find frequent items from a data stream, including not only developing methods but also comparing them [5, 19]. Here, we classify them into three categories as follows.

**Sampling based Approach.** This approach involves a probabilistic process, and obtained items are random samples over all the items occurring in a data stream. Vitter [24] proposed a uniform sampling method from a data stream by which, in essence, higher frequency items are sampled much more than lower ones. Improving the space requirement for the sampling, Gibbons and Matias [11] invented a method called concise sampling which efficiently represents sampled items. By slightly modifying the method, they also proposed counting sampling to estimate the frequency of each item more accurately. A similar approach was adopted in [20] to obtain high frequency items.

**Counter based Approach.** A very basic form of the counter based approach is MAJORITY that finds the majority item in a stream if it exists [2, 10]. By generalizing MAJORITY, Misra and Gries [23] developed a method to find items occurring at least $N/k$ times and it was improved in per-item processing time by [9, 15]. LOSSYCOUNTING [20] does the same job but it additionally guarantees that no item whose count is less than $N(1/k-\epsilon)$ is reported. SPACESAVING [22] reduces the space requirement not only for a general data distribution but also for a Zipf distribution.

**Sketch based Approach.** The sketch based approach is usually based on using multiple hash functions to map incoming items to a hash table. This can be also understood as maintaining a list of independent counters where each counter is shared by a few items and the sharing is determined by the hash functions. Charikar et al. proposed COUNTSKETCH that computes items appearing at least $N/(k+1)$ times with probability $1-\delta$ while requiring $O(k/\epsilon^2 \log N/\delta)$ memory spaces. The space requirement was improved by COUNTMIN [6]. GROUPTEST [7] was developed for a hot item query, which groups items and assumes one frequent item in each group. Jin et al. improved GROUPTEST in space and their algorithm guarantees the minimum count of items outputted [14].

## 2.2 Finding Recent Frequent Items

Despite many algorithms to find frequent items from a data stream, researchers have agreed that recent items are more important than old ones and answering a query of finding recently frequent items is often required. Below, we introduce two approaches for the purpose.

**Sliding Window based Approach.** This approach divides recent times into window blocks, performs counting for items in the windows, and aggregates them. Golab et al. [12] proposed a method based on basic window blocks for identifying frequent items in packet streams in which item frequency is known to follow a power-law distribution. They also extended the work to the more general case that the item frequency follows a multinomial distribution [13]. There have been works using windows of various sizes. Arasu and Manku [1] provided deterministic and randomized algorithms for $\epsilon$-approximate quantiles over sliding windows. Dallachiesa and Palpanas [8] studied the problem in ad hoc time windows. They use overlapping window blocks whose sizes exponentially grow by which a query for frequent items during a certain recent time period can be answered more accurately.

**Time-aware based Approach.** This approach implicitly considers the recentness. Liu et al. [18] proposed a pruning method, a key operation of counter based algorithms, considering time information so that an older item is more likely to be pruned than a more recent one when the memory becomes full. A similar approach has been examined in [4, 25, 26]. All of them adopt a time fading factor in developing methods to find frequent items which decreases a weight of an item over time. As a result, recent items have more weights, leading to more accurate results. Our proposed algorithms in this paper also belong to this category. We show that time-weighted counting can be done via sampling which guarantees its accuracy in expectation, and propose our main algorithm TWMINSWAP via derandomization.

# 3. PROPOSED METHODS

**Algorithm 1:** TWSAMPLE: Randomized Time-Weighted Counting

**Input**: A data stream $S$, a number $k$ of counters, a decaying parameter $\alpha$, and increase ratio $\sigma$ of the penalty term.

```
λ ← 1.
K ← ∅.
foreach new item u from S do
    Downsampling(α).
    if bernoulli(α^{λ−1}) = 1 then
        if u ∈ K then
            c_u ← c_u + 1.
        else
            K ← K ∪ {u}.
            c_u ← 1.
        end
        while |K| > k do
            λ ← λ + σ.
            Downsampling(α^σ).
        end
    end
end
Subroutine Downsampling(θ)
foreach v ∈ K with counter c_v do
    c_v ← binomial(c_v, θ).
    if c_v = 0 then
        K ← K\{v}.
    end
end
```

In this section, we propose our method for finding time-weighted top-$k$ items from a data stream. The main idea is derandomization of a sampling based algorithm. As a result, we propose a deterministic algorithm TWMINSWAP, which requires only $O(k)$ memory spaces where $k$ is the number of time-weighted frequent items that we want to find. To develop TWMINSWAP, we first propose a sampling based randomized algorithm TWSAMPLE whose performance is guaranteed in expectation. However, the probabilistic nature of TWSAMPLE requires several independent sampling sessions to achieve high accuracy, leading to large memory spaces and slow running time. On the other hand, TWMINSWAP achieves high accuracy with fast running time while using only a single session.

We start with the definition a time-weighted count of an item [4, 25, 26].

DEFINITION 1 (TIME-WEIGHTED COUNT). *Let $u$ be an item occurring in a data stream at times $t_1, \ldots, t_c$. The time-weighted count of the item $u$ is defined by*

$$W(u) = \sum_{i=1}^{c} \alpha^{t_{cur}-t_i},$$

*where $\alpha$ is a decaying parameter and $t_{cur}$ is the current time.*

## 3.1 Randomized Algorithm

To develop a sampling based randomized algorithm, we first define a penalized time-weighted count as follows.

Definition 2 (Penalized Time-weighted Count). *Let $u$ be an item occurring in a data stream and let $T_u$ be a set of the times at which $u$ has occurred. The penalized time-weighted count of the item $u$ is defined by*

$$P(u) = \sum_{t \in T_u} \alpha^{t_{cur}-t+\lambda-1},$$

*where $\alpha$ is a decaying parameter, $t_{cur}$ is the current time, and $\lambda$ is a default penalty term for items just arriving.*

Let the sequence of items till time $t_{cur}$ be $u_1, \ldots, u_{t_{cur}}$. Given $0 < \alpha \leq 1$ and $\lambda \geq 1$, our randomized algorithm samples each item $u_t$ with probability $\alpha^{t_{cur}-t+\lambda-1}$. This is incrementally done with increasing $\lambda$ over time to ensure that the number of distinct items in the samples is at most $k$. Indeed, our algorithm can be understood as an extension of the uniform sampling in a data stream [11] to a time-weighted sampling. Below, we call that $u$ is monitored if $u$ has been sampled at least once.

Precisely, our randomized algorithm TwSample requires three parameters: the maximum number $k$ of monitored items, the time-weighting factor $\alpha$, and the increase ratio $\sigma$ for the penalty term $\lambda$. Also there are three pieces of information incrementally updated: the penalty term $\lambda$, the set $K$ of monitored items, and counters $c_v$ associated with each $v \in K$. Initially, $\lambda = 1$ and $K = \emptyset$. A new item $u$ currently arriving is determined whether sampled or not with probability $\alpha^{\lambda-1}$. The sampling is done as follows: if $u$ is currently monitored, i.e. $u \in K$, $c_u$ is incremented by 1; otherwise $u$ is added to $K$ with its associated counter $c_u = 1$. After the sampling, if $|K| > k$, *downsampling* is applied to all the sampled items so far with increasing $\lambda$. Precisely, $\lambda$ is incremented by $\sigma$, and for each $v \in K$, $c_v$ is updated by a random number drawn from $binomial(c_v, \alpha^{\sigma})$[1]. If $c_v$ becomes 0, $v$ is evicted from $K$. This downsampling is repeated until $|K| \leq k$. Lastly, whenever one time step passes, all items in $K$ are unconditionally downsampled as follows: for each $v \in K, c_v = binomial(c_v, \alpha)$. Algorithm 1 fully describes TwSample.

**Effect of $\sigma$.** Essentially, $\sigma$ determines the sampling rate for evicting existing items so that the number of sampled items does not exceed $k$. As $\sigma$ gets smaller, the amount of decrements of each item count in Line 14 is reduced, leading to longer time for the eviction process in Line 12 to 15. Accuracy may increase since we do not evict more than required. On the other hand, as $\sigma$ gets larger, the eviction process finishes quickly, although the accuracy may decrease since we may evict more than one item.

Lemma 1. *At time $t_{cur}$ with the penalty term $\lambda$, each item $u$ occurring at time $t \leq t_{cur}$ has been sampled with probability*

$$\Pr[u \text{ is sampled}] = \alpha^{t_{cur}-t+\lambda-1}.$$

Proof. Let $u$ be a new item at $t = 1$ and $t_{cur} = 1$. It is unconditionally sampled because $\lambda = 1$, and all counters are empty. In other words, $u$ is sampled with the probability $\alpha^{t_{cur}-t+\lambda-1} = 1$.

Let $t_{cur} \geq 1$. Assume that the lemma holds for time $t_{cur}$. That is, for each sample $v$ at time $t \leq t_{cur}$, it has been sampled with probability $\alpha^{t_{cur}-t+\lambda-1}$. Let us consider the process for $t_{next} = t_{cur} + 1$. In Line 4, all samples are downsampled with probability $\alpha$. This means that after Line 4, remaining samples are with probability $\alpha^{t_{cur}+1-t+\lambda-1}$ which matches the sampling probability at time $t_{next} = t_{cur} + 1$.

Next, we verify from Line 5 to 11. Let $u$ be a new item occurring at $t = t_{next}$. Clearly, $u$ is sampled with probability $\alpha^{t_{next}-t+\lambda-1} = \alpha^{\lambda-1}$ by Line 5, regardless of whether $u$ is currently monitored or not. Let us verify the downsampling. Let $d$ be the number of iterations by the while statement in Line 12. Then, after the downsampling, each sample experiences re-sampling with probability $\alpha^{d\sigma}$, and thus each sample is with probability $\alpha^{t_{next}-t+\lambda+d\sigma-1}$ which matches the update of $\lambda = \lambda + d\sigma$.

Note that the sampling probability does not increase over time since $\lambda$ increases or remains the same at every time step. Hence, an item that is not a sample at a certain time cannot be a sample in the future. □

The following corollary is directly derived from Lemma 1.

Corollary 1. *At any time $t_{cur}$ in* TwSample*, the expectation of a counter $c_u$ of a monitored item $u \in K$ is*

$$\mathbb{E}[c_u] = P(u) = \sum_{t \in T_u} \alpha^{t_{cur}-t+\lambda-1},$$

*where $T_u$ is a set of times at which $u$ has occurred.*

Since we know $\lambda$ at any time, we can compute the expected time-weighted count for a monitored item $u$. That is, the estimated time-weighted count of $u \in K$ becomes $\alpha^{1-\lambda} c_u$.

Lemma 2. *At any time, the probability $p_u$ that an item $u$ is monitored satisfies the following inequality:*

$$p_u \geq 1 - \exp(-P(u)/2).$$

Proof. Note that $p_u = \Pr[c_u > 0]$ since $c_u = \sum_{i=1}^{|T_u|} c_u(i)$ is a random variable where $c_u(i)$ indicates whether the $i$-th occurrence of $u$ is sampled or not. Applying the Chernoff bound, we obtain the following inequality:

$$p_u = \Pr[c_u > 0] = 1 - \Pr[c_u = 0] \geq 1 - \exp(-\mathbb{E}[c_u]/2).$$

Since $\mathbb{E}[c_u] = P(u)$ by Corollary 1, the proof is done. □

Lemma 2 states that as an item is more insignificant, the probability that it is not monitored increases exponentially.

Although TwSample is simple and provides the theoretical guarantees of its output, its performance may be degraded due to its probabilistic nature. First, the running time may become slow because the number of iterations for the downsampling with increasing $\lambda$ is not fixed and the time for drawing random variables from $binomial(c, \theta)$ used in the downsampling depends on $c$. Second, discovered top-$k$ items and the associated counters may be inaccurate due to unintendedly large $\lambda$. This inaccuracy can be resolved by maintaining $s$ number of independent sessions each of which monitors at most $k$ items, but it leads to more memory spaces and longer running time. In the next section, we propose a deterministic variation of TwSample, which is fast and requires a single session of monitored items of size at most $k$.

## 3.2 Deterministic Algorithm

In this section, we propose TwMinSwap for efficient top-$k$ time-weighted frequent items discovery. This algorithm

[1]Here, $binomial(\omega, \theta)$ denotes a binomial random variable with the number $\omega$ of independent trials and the success probability $\theta$.

**Algorithm 2:** TwMinSwap: Deterministic Time-Weighted Counting

**Input**: A data stream $S$, the number of counters $k$, and a decaying parameter $\alpha$.

$K \leftarrow \emptyset$.
$t_{cur} \leftarrow 0$.
**foreach** *new item* $u$ *from* $S$ **do**
  $t_{cur} \leftarrow t_{cur} + 1$.
  **foreach** $v \in K$ **do**
    $c_v \leftarrow c_v \times \alpha$.
  **end**
  **if** $u \in K$ **then**
    $c_u \leftarrow c_u + 1$.
  **else if** $|K| < k$ **then**
    $K \leftarrow K \cup \{u\}$.
    $c_u \leftarrow 1$.
  **else**
    $v^* \leftarrow \mathrm{argmin}_{v \in K} c_v$.
    **if** $c_{v^*} < 1$ **then**
      $K \leftarrow K \backslash \{v^*\} \cup \{u\}$.
      $c_u \leftarrow 1$.
    **end**
  **end**
**end**

is a deterministic version of TwSample with truncating insignificantly old items.

The main idea is to record the expected number of samples for each item directly instead of applying the random process. Concretely, for an item $u$ occurring at $t \leq t_{cur}$, instead of incrementing $c_u$ by 1 with probability $\alpha^{t_{cur}-t}$, we increment $c_u$ by $\alpha^{t_{cur}-t}$ with probability 1. Then, the downsampling with rate $\theta$ becomes that for each monitored item $v \in K$, $c_v = \theta c_v$. With this deterministic scenario, however, we encounter a problem when all counters become full. Precisely, because the expected count for an item occurring at least one time in the stream never becomes 0, it needs pruning for dropping an insignificant monitored item to start monitoring a new item. We propose a simple heuristic for the pruning which does not require additional memory spaces, leading to smaller memory usage compared with the previous approaches [4, 25, 26]. We note that this proposed pruning method here corresponds to decreasing the sampling rate by increasing $\lambda$ in TwSample.

Details of the heuristic for the pruning are as follows. Let $K$ be a set of currently monitored items where $|K| = k$, and $u$ be an item just arriving from a data stream. Our pruning method first finds an item $v^* \in K$ having the minimum time-weighted count $c_{v^*} = \min_{v \in K} c_v$. If $c_{v^*} < 1$, we drop $v^*$ and start monitoring $u$ with initial count 1; otherwise, $u$ is ignored. This swapping of $u$ and $v^*$ makes sense because $c_{v^*}$ is computed by a few most recent occurrences of $v^*$ but smaller than the effect 1 by the single occurrence of $u$ at $t_{cur}$. The overall procedure of TwMinSwap is described in Algorithm 2.

Advantages of TwMinSwap are summarized as follows. First, its memory usage is small. For each item, only an item identifier and its time-weighted count are maintained. This simple structure enables to reduce per-item computation compared with similar counter based approaches [25], and greatly saves memory spaces compared with sketch based algorithms [4]. Second, TwMinSwap requires the minimal parameters: $k$ and $\alpha$. This especially gives the benefit of reducing efforts for parameter tuning in practice. Third, in contrast to TwSample, TwMinSwap guarantees to output $k$ number of items so long as $N \geq k$.

**Per-item Processing Time.** The main time-consuming operations are: (1) computing an item with the minimum count, and (2) multiplying $\alpha$ to all counters each of which requires scanning $K$. The second operation can be eliminated by maintaining the most recent time for each item when it occurs [25]. In this way, we benefit in speed when a new item is already monitored. In contrast, the first operation taking $O(k)$ time is unavoidable. Although an efficient data structure was proposed for the case without time-weighting [9], it cannot be applied to our time-weighted case since our counters record real numbers with which difference between two numbers is unfixed. As a result, TwMinSwap requires $O(k)$ computation for each iteration.

### 3.2.1 Analysis

We analyze TwMinSwap especially for the condition that a monitored item is not evicted from $K$. More precisely, Lemmas 3 and 4 state that TwMinSwap will not evict items whose frequencies of occurrences are above certain thresholds for a general and a power-law item distribution cases, respectively.

Lemma 3. *Let $0 < \alpha \leq 1$ be a time-decaying parameter of* TwMinSwap. *Any item with count $c \geq 1$ will not be evicted from $K$ if it occurs at least once per every $1 - \log_\alpha \gamma$ times where $\gamma = \min\{1 + \alpha, c\}$.*

Proof. Assume that $v \in K$ with count $c_v = c \geq 1$ at a certain time occurs for every $d$ times. Note that any item whose count is at least 1 is never evicted from $K$ by construction. Let us define the following function.

$$g(r+1) = (g(r)\alpha + 1)\,\alpha^{d-1},$$

where $g(1) = c\alpha^{d-1}$. It is clear that $g(r)$ is $c_v$ after $rd - 1$ time steps. Since $c_v$ always decreases from $(r-1)d + 1$ to $rd - 1$, it suffices to show that $g(r) \geq 1$ with $d \leq 1 - \log_\alpha \gamma$ for every $r \geq 1$ where $\gamma = \min\{1 + \alpha, c\}$.

For $r = 1$, assume that $c \leq 1 + \alpha$; then,

$$g(1) = c\alpha^{d-1} \geq c\alpha^{-\log_\alpha c} = 1.$$

Assume that $c > 1 + \alpha$.

$$g(1) = c\alpha^{d-1} > (1+\alpha)\alpha^{d-1} \geq (1+\alpha)\alpha^{-\log_\alpha(1+\alpha)} = 1.$$

For $r > 1$, assume that $g(r-1) \geq 1$; then,

$$\begin{aligned} g(r) &= (g(r-1)\alpha + 1)\,\alpha^{d-1} \geq (\alpha+1)\alpha^{d-1} \\ &\geq \min\{1+\alpha, c\} \times \alpha^{-\log_\alpha \min\{1+\alpha, c\}} = 1. \end{aligned}$$

□

Below, we show which item is expected not to be evicted from $K$ before the next occurrence of the item for a power-law item distribution.

Lemma 4. *Let $n$ be the number of items and let us consider a power-law item distribution with the exponent 2. Any monitored item $i \leq \sqrt{(1 - \log_\alpha(1+\alpha))/1.7}$ with count $c_i \geq \alpha^{1-1.7i^2}$ is expected not to be evicted.*

**Table 3: Top-5 items with and without time-weighting for the two types of item distributions. With the dynamic item distribution, frequent items are different from those for the static distribution. Especially, with time-weighting, frequent items are completely different between the static and the dynamic distributions—items *recently* occurring many times are placed in high ranks for the dynamic distribution.**

| | | No time-weighting | | Time-weighting | |
|---|---|---|---|---|---|
| | Rank | Static | Dynamic | Static | Dynamic |
| (high) | 1 | 1 | 1 | 1 | 10000 |
| | 2 | 2 | 2 | 2 | 9999 |
| | 3 | 3 | 3 | 5 | 9998 |
| | 4 | 4 | 10000 | 4 | 9995 |
| (low) | 5 | 5 | 4 | 7 | 9997 |

PROOF. For an item $i \in [1, n]$, the probability that it occurs in a stream is

$$\Pr[i \text{ occurs}] = i^{-2}/Z,$$

where $Z$ is the normalization constant. Hence, the expected number of occurrences before $i$ occurs becomes $i^2 Z$. Since an occurrence of $i$ obeys $geometric(i^{-2}/Z)$, the probability that $i$ does not occur during $i^2 Z$ time steps is less than $1/e$.

Assume that $c_i \leq \alpha + 1$. The following guarantees in expectation that $i$ will not be evicted by Lemma 3:

$$1 - \log_\alpha c_i \geq i^2 Z \iff 1 - i^2 Z \geq \log_\alpha c_i \iff \alpha^{1-i^2 Z} \leq c_i.$$

The feasible $i$ should satisfy

$$\alpha^{1-i^2 Z} \leq \alpha + 1 \iff i \leq \sqrt{\frac{1 - \log_\alpha(1+\alpha)}{Z}}.$$

Assume $c_i > \alpha + 1$. The following inequality guarantees in expectation that $i$ will not be evicted by Lemma 3:

$$1 - \log_\alpha(1+\alpha) \geq i^2 Z \iff i \leq \sqrt{\frac{1 - \log_\alpha(1+\alpha)}{Z}}.$$

Finally, $Z \leq \sum_{j=1}^{\infty} j^{-2} = \pi^2/6 \leq 1.7$, which completes the proof. □

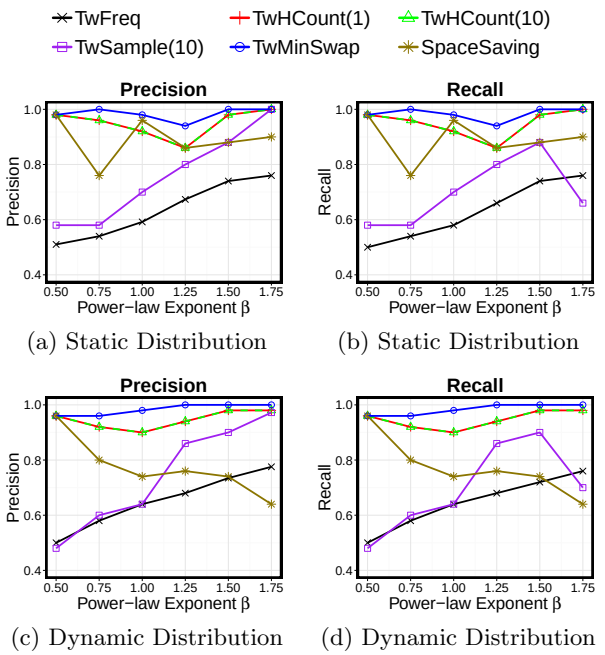
# 4. EXPERIMENTS

In this section, we present experimental results to show the performance of our proposed TWMINSWAP with synthetic data streams. Especially, we want to answer the following questions:

- Q1 How many top-$k$ time-weighted frequent items can we discover?
- Q2 How accurately can we estimate time-weighted counts of the discovered items?
- Q3 How fast is TWMINSWAP?

## 4.1 Setup

We consider two types of data streams generated from power-law distributions to simulate bursty item occurrences where $N$ is the stream length and $n$ is the number of distinct items.




(a) Static Distribution (b) Static Distribution

(c) Dynamic Distribution (d) Dynamic Distribution

**Figure 2: Our proposed TWMINSWAP shows the best performance in both precision and recall whose values are close to 1, regardless of the distribution types and their $\beta$ values. For distributions with small $\beta$ in which frequencies of items are relatively even, TWSAMPLE shows low precision and recall, but as $\beta$ gets larger, the values rapidly increase. TWFREQ also shows better performance for large $\beta$, but the improved precision and recall are limited below 0.8. Regardless of the hash table size, TWHCOUNT shows the second best performance; however the performance is below that of TWMINSWAP. As expected, SPACESAVING is degraded for the dynamic item distributions while performing well on average for the static item distributions.**

- Static distribution: $N$ items are generated from the following power-law distribution.

$$\Pr[i \text{ is generated}] \propto i^{-\beta},$$

where $i \in [1, n]$.
- Dynamic distribution: for $0 \leq r \leq 1$, the first $rN$ items are generated from

$$\Pr[i \text{ is generated}] \propto i^{-\beta},$$

and the last $(1-r)N$ items are generated from

$$\Pr[i \text{ is generated}] \propto (n - i + 1)^{-\beta},$$

where $i \in [1, n]$. This provides the setting that frequent items differ from time-weighted frequent items.

Table 3 shows the difference between the two types of distributions.

In our experiments, $\beta$ is varied in $\{0.5, 0.75, 1, 1.25, 1.5, 1.75\}$; $N = 10^6, n = 10^4, r = 0.8$ and $k = 50$ are fixed. Here, as $\beta$ gets larger, item frequencies get skewed more. Note that although the stream length $N$ is fixed in our experiments, the per-item time and the space complexities of our algorithms are independent on $N$.

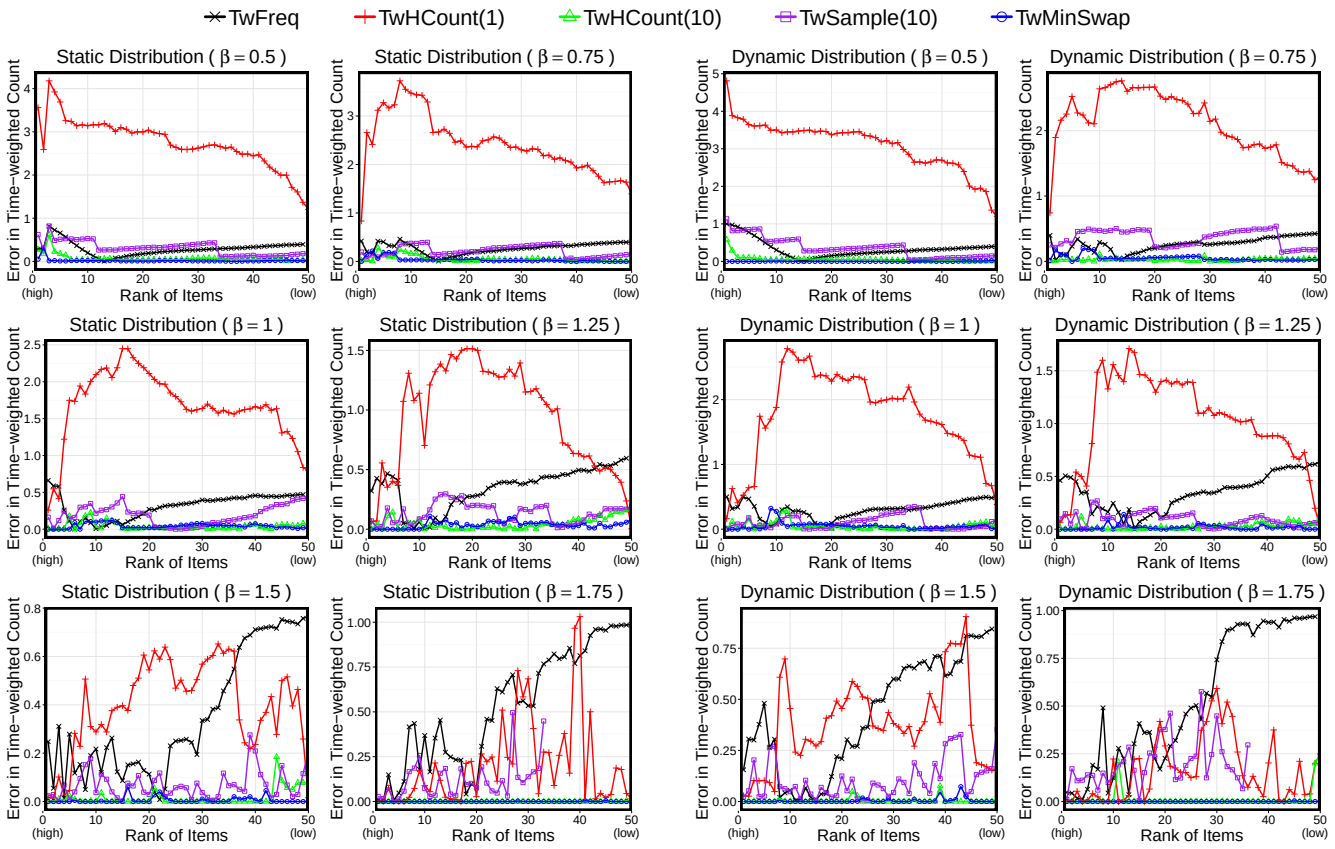



**Figure 3: The estimated time-weighted counts by TwMinSwap are the most accurate. The plots show the error in estimated time-weighted counts of the algorithms for at most $k$ number of discovered items (the lower, the better). While TwHCount(10) shows the second best performance, TwHCount(1) using smaller memory spaces performs poorly especially for small $\beta$. The accuracy of TwFreq is generally better for high rank items than for low rank items. TwSample(10) shows slightly better performance than TwFreq on average, but in contrast to TwFreq, there is no great degradation of the estimations for low rank items in TwSample(10).**

We consider the following competitors, and in our experiments, all methods are implemented in Java.

- TwFreq [25]: a counter based algorithm to find time-weighted frequent items from a data stream.
- TwHCount [4]: a sketch based algorithm to find time-weighted frequent items from a data stream.
- SpaceSaving [22, 5]: a counter based algorithm to find frequent items from a data stream. Since this is without time-weighting, we compare this algorithm with the others only in precision and recall.

The memory requirements for all the algorithms are as follows. TwMinSwap, TwFreq, and SpaceSaving require $O(k)$ memory spaces; TwSample requires $O(sk)$ where $s$ is the number of parallel sessions each of which independently samples at most $k$ distinct items; TwHCount requires $O(k + rm)$ where $r$ is the number of hash functions and $m$ is a range size of the hash functions.

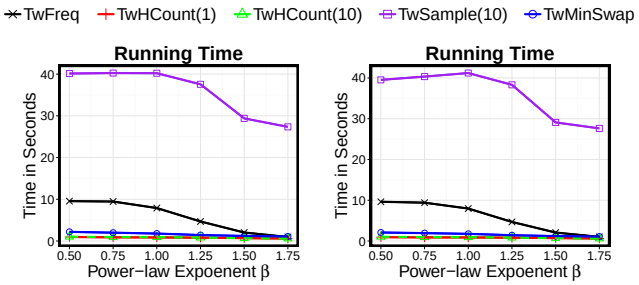
We denote TwSample with $s$ number of the independent sessions by TwSample($s$), and TwHCount with the hash table size $w$% of $n$, i.e. $rm = \frac{wn}{100}$, by TwHCount($w$). For TwSample, we use $s = 10$ and $\sigma = 0.0001$; for TwHCount, we use the parameters in the original paper [4], and set hash table sizes $rm$ to 1% and 10% of $n$.

The overall comparison is summarized in Table 1 and Figure 1. TwMinSwap outperforms the others in terms of accuracy and memory usage, and its speed is comparable to that of the fastest competitor TwHCount.

## 4.2 Discovering Top-$k$ Time-weighted Frequent Items

Figure 2 presents accuracy of discovered items in terms of precision & recall. Overall, our proposed TwMinSwap outperforms other algorithms regardless of the types of item distributions and the exponent values $\beta$. Its precision and recall are always very close to 1. TwSample(10) improves precision and recall as $\beta$ gets larger in general. The reason why the recall of TwSample(10) is low for $\beta = 1.75$ is that a large amount of probability density is assigned to only fewer items as $\beta$ gets larger, leading to a small number $\hat{k} < k$ of discovered items. The exact numbers are 33 and 36 for the static and the dynamic distributions, respectively. For TwMinSwap and TwHCount, always $\hat{k} = 50$, and for TwFreq, always $\hat{k} \geq 49$.

TwFreq shows a similar pattern to TwSample(10) but its improvement is less significant than TwSample(10). TwHCount results in high precision and recall regardless of hash table sizes but they are still below TwMinSwap. Space-




(a) Static Distribution (b) Dynamic Distribution

**Figure 4: TwMinSwap is faster than TwSample(10) and TwFreq in most cases of $\beta$. Although TwHCount is slightly faster than TwMinSwap, its memory usage is much larger than TwMinSwap as shown in Figure 1.**

Saving performs quite well for the static item distributions: both precision and recall are about 0.9 on average. However, since SpaceSaving does not consider a time-weighting factor, for the dynamic item distributions its performance is greatly degraded as $\beta$ gets larger.

The overall result implies that our time-weighted counting plays an important role in finding recent frequent items from data streams.

## 4.3 Estimating Time-weighted Counts

Accurately estimating time-weighted counts for discovered items enables to quantitatively compare them. Figure 3 shows estimated time-weighted counts of the discovered top-$k$ items. Notably, TwMinSwap estimates the true time-weighted counts very accurately, which is shown by the blue line (almost overlapped with green).

Since TwFreq provides an upper and a lower bounds of the true time-weighted count, we choose the mean of the two bounds. In general, as ranks of items get lower, its accuracy is generally degraded. One reason of the poor estimation of TwFreq for low rank items is that TwFreq is originally developed to find frequent items having time-weighted counts above a certain threshold rather than top-$k$ ones though it maintains at most $k$ items. TwSample(10) shows the third best performance, and estimates time-weighted counts more accurately for relatively large $\beta$. The performance is slightly better than TwFreq on average, but TwSample(10) is not degraded for relatively low rank items.

The performance of TwHCount highly depends on the hash table size $rm$. The estimation of TwHCount(1) results in larger error while TwHCount(10) is comparable to TwMinSwap. This is notable because the estimation of TwMinSwap is as accurate as that of TwHCount(10) which keeps an additional hash table of size $0.1n$ for accuracy. The comparison of the error on average is shown in Figure 1.

## 4.4 Running Time

Figure 4 shows running times of the algorithms taken to process all the items in the data streams. The overall trend is that running time decreases over increasing $\beta$. This is because with large $\beta$, the probability of a new item being already monitored increases, leading to infrequent invoking eviction processes such as *DownSampling* in TwSample. TwSample(10) shows the slow running times due to performing 10 independent sampling.

Although TwFreq has the same per-item processing time as TwMinSwap in the big-O notation, TwFreq involves more computations for the eviction process than TwMinSwap. TwFreq updates more variables and scans monitored items twice while TwMinSwap scans them once. As a result, the running time of TwFreq changes more dramatically than TwMinSwap.

Since TwHCount has $O(r)$ per-item processing time, it is the fastest. In fact, this fast running time is achieved by maintaining the hash table of size $rm$ for approximate time-weighted counting, which leads to more memory spaces than TwMinSwap. Although the running time of TwHCount does not depend on $m$, to guarantee small error of estimated time-weighted counts of discovered items, $rm$ should be large as shown in Figure 3. In our experiments, $rm = 0.1n$ is satisfactory while $rm = 0.01n$ is not.

All algorithms show no meaningful difference in running time with respect to the two types of distributions.

# 5. DISCOVERY

In this section, we present discoveries from applying TwMinSwap to several real world data streams.

## 5.1 MemeTracker Dataset

**Setup.** The MemeTracker dataset [16] provides quotes and phrases from blogs and news media. We consider a keyword stream consisting of words in the quotes and the phrases where 571 stopwords provided in [17] are excluded. The stream covers time period between Aug 2008 and Apr 2009; the length of the stream is 1,681,760,809; we consider one minute as one time step.

**Results.** We run TwMinSwap with $k = 300$ and $\alpha = 0.9$, and examine top-300 keywords for every month.

Figure 5a shows the tracking results of keywords related to the U.S. presidential election in Nov 4 2008. The values for each month are normalized time-weighted counts divided by the sum of those of $k$ number of discovered items. Since multiple items can occur at one time step, this normalization is required to eliminate effects of undesirably large time-weighted counts due to relatively large stream lengths for certain time periods. Both keywords related to the candidates *obama* and *mccain* were mentioned actively before, and received less attentions after the election. Notably, despite high frequencies of both keywords, the winner *obama* was more frequently mentioned in blogs and media than the loser *mccain*. Even after the election, *obama* occasionally becomes hot.

Figure 5b and Figure 5c show sudden arising and quick vanishing of keywords closely related to two incidents: the Mumbai terror attack in Nov 2008, and the Gaza War beginning on Dec 2008. Although each incident happened in the last part of a month, TwMinSwap correctly detects related keywords as hot items in the report for that month.

## 5.2 Amazon Movie Review Dataset

**Setup.** The Amazon movie review dataset [21] provides user reviews with product id information where the movie title of each product id can be checked by `http://www.amazon.com/dp/PRODUCT_ID`. We consider the stream of the product ids. The stream covers the period from Aug 20 1997

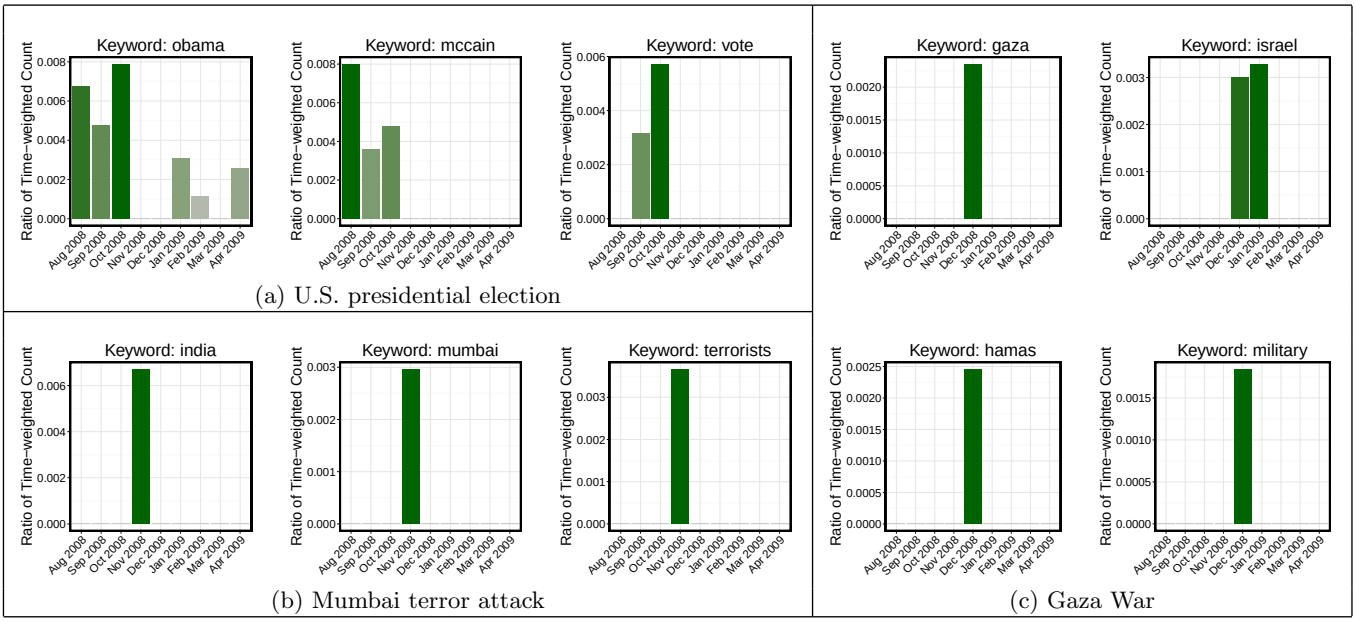



**Figure 5: (a) Changes of time-weighted counts of keywords related to the presidential election in Nov 4 2008. The keywords *obama, mccain* and *vote* become hot keywords before and cooled down after the election. Notably, the winner *obama* is more actively mentioned than the loser *mccain* before the election. Also, the winner *obama* does not disappear after the election in contrast to the loser *mccain*. (b) Changes of time-weighted counts of keywords related to the Mumbai terror attack in Nov 26 2008. As similar to the pattern for the Gaza War, keywords related to the Mumbai terror attack such as *india, mumbai* and *terrorists* show sudden rises right after the attack time. (c) Changes of time-weighted counts of keywords related to the Gaza War beginning in the last part of Dec 2008. In December, several keywords related to the war suddenly had arisen and quickly disappeared. Note that the war ended in the middle part of January.**

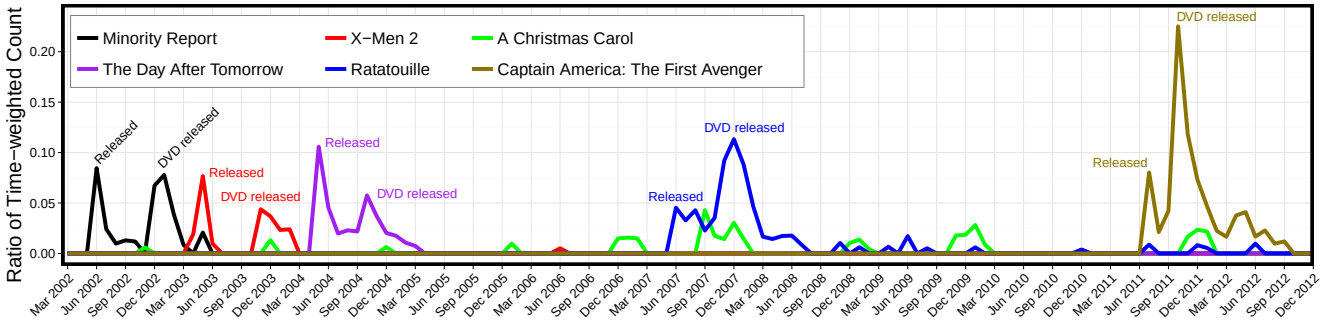



**Figure 6: Changes of time-weighted counts of movies reviewed by users in Amazon. We observe two patterns. (1) The general pattern is doubly-active attention to movies when they are released at theaters and in DVDs. (2) The minor pattern is periodical attention: e.g., *A Christmas Carol* is popular in every winter.**

to Sep 25 2012; the length of the stream is 7,911,684; the number of distinct product ids is 253,059.

**Results.** We run TwMinSwap with $k = 100$ and $\alpha = 0.9$. Figure 6 shows the tracking result of several movies among the top-100, which is summarized as two patterns. The major pattern is doubly-active attention when a movie is released at theaters and in DVD as for *Minority Report, X-Men 2, The Day After Tomorrow, Ratatouille*, and *Captain America: The First Avenger*. The other pattern is periodical attention: e.g. *A Christmas Carol* appears in every winter.

## 5.3 Yelp Dataset

**Setup.** The Yelp dataset[2] provides tip data for businesses by users. Here, the tips are short comments about the businesses, and each business has several associated categories. We consider the stream of the business ids. The stream covers the period from Apr 16 2009 to Feb 11 2014; the length of the stream is 113,993; the number of distinct businesses is 15,585.

**Results.** We run TwMinSwap with $k = 50$ and $\alpha = 0.9$, and track the top-30 hot businesses per month. For each month, we obtain a category distribution with respect

[2] http://www.yelp.com/dataset_challenge/

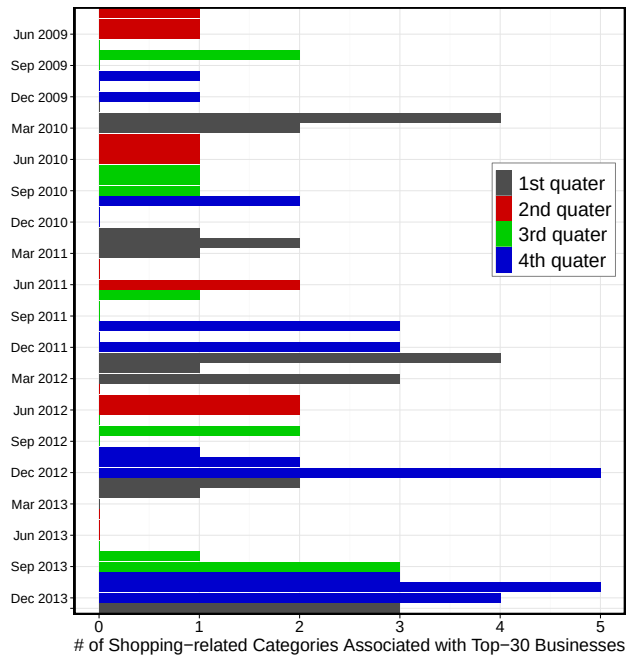



**Figure 7: Count distribution over time for shopping related categories—*Shopping* and *Shopping Centers*. For each month, the value is calculated with respect to top-30 businesses. In winter, there were a number of visits to shopping businesses.**

to the top-30 businesses. Figure 7 shows the result for shopping-related categories—*Shopping* and *Shopping Centers*. Around the new years, shopping activity increased. This reflects that there are several special days such as Christmas, New Year, and Valentine Day in winter.

# 6. CONCLUSION

In this paper we propose TwMinSwap, a fast, accurate, and space-efficient method for tracking recent frequent items from high speed data streams. We also present interesting discoveries from real world streams. The main contributions are the following:

- We propose TwMinSwap for efficient time-weighted counting of frequent items in data streams. TwMinSwap is inspired by TwSample, our sampling based randomized algorithm with nice theoretical guarantees. Both methods require only $O(k)$ memory for tacking top-$k$ items.
- Conducting extensive experiments on synthetic data streams, we show that TwMinSwap is fast, and outperforms all existing methods in accuracy.
- We analyze real world data streams, and we discover interesting patterns including the difference of trends between the winner and the loser of the U.S. presidential election, and doubly-active pattern of movies.

## Acknowledgments

This research was supported by KT Institute of Convergence Technology, and by the KAIST High Risk High Return Project (HRHRP).

# 7. REFERENCES

[1] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *PODS*, 2004.
[2] R. S. Boyer and J. S. Moore. Mjrty: A fast majority vote algorithm. In *Automated Reasoning: Essays in Honor of Woody Bledsoe*, 1991.
[3] J. H. Chang and W. S. Lee. Finding recent frequent itemsets adaptively over online data streams. In *KDD*, 2003.
[4] L. Chen and Q. Mei. Mining frequent items in data stream using time fading model. *Information Sciences*, 257(0):54–69, 2014.
[5] G. Cormode and M. Hadjieleftheriou. Methods for finding frequent items in data streams. *The VLDB Journal*, 19(1):3–20, 2010.
[6] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. In *LATIN*, 2004.
[7] G. Cormode and S. Muthukrishnan. What's new: Finding significant differences in network data streams. In *INFOCOM*, 2004.
[8] M. Dallachiesa and T. Palpanas. Identifying streaming frequent items in ad hoc time windows. *Data Knowl. Eng.*, 87:66–90, 2013.
[9] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *ESA*, 2002.
[10] M. J. Fischer and S. L. Salzberg. Finding a majority among $n$ votes: Solution to problem 81-5 (Journal of Algorithms, june 1981). *Journal of Algorithms*, 3(4):362–380, 1982.
[11] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD*, 1998.
[12] L. Golab, D. DeHaan, E. D. Demaine, A. López-Ortiz, and J. I. Munro. Identifying frequent items in sliding windows over on-line packet streams. In *Internet Measurement Comference*, 2003.
[13] L. Golab, D. DeHaan, A. López-Ortiz, and E. D. Demaine. Finding frequent items in sliding windows with multinomially-distributed item frequencies. In *SSDBM*, 2004.
[14] C. Jin, W. Qian, C. Sha, J. X. Yu, and A. Zhou. Dynamically maintaining frequent items over a data stream. In *CIKM*, 2003.
[15] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28:51–55, 2003.
[16] J. Leskovec, L. Backstrom, and J. Kleinberg. Meme-tracking and the dynamics of the news cycle. In *KDD*, 2009.
[17] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.
[18] H. Liu, Y. Lu, J. Han, and J. He. Error-adaptive and time-aware maintenance of frequency counts over data streams. In *WAIM*, 2006.
[19] N. Manerikar and T. Palpanas. Frequent items in streaming data: An experimental evaluation of the state-of-the-art. *Data Knowl. Eng.*, 68(4):415–430, 2009.
[20] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, 2002.
[21] J. J. McAuley and J. Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *WWW*, 2013.
[22] A. Metwally, D. Agrawal, and A. E. Abbadi. Efficient computation of frequent and top-k elements in data streams. In *ICDT*, 2005.
[23] J. Misra and D. Gries. Finding repeated elements. *Science of Computer Programming*, 2(2):143 – 152, 1982.
[24] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.
[25] S. Zhang, L. Chen, and L. Tu. Frequent items mining on data stream based on time fading factor. In *AICI*, 2009.
[26] S. Zhang, L. Chen, and L. Tu. Frequent items mining on data stream using hash-table and heap. In *ICIS*, 2009.