Estimating Trending Topics on Twitter with Small Subsets of the Total Data

Albert Lee, Evan Miller, and Kiran Vodrahalli

January 7, 2015

1 Introduction

2 Problem

3 Previous Work

4 Algorithm

4.1 Description

We separate the problem of finding trending topics on Twitter into two parts. First, we need to maintain a data structure that efficiently stores data about all occurrences of every hashtag seen in the past. We also maintain a separate data structure that allows us to quickly gather information about the most recent hashtags seen.

We want the former data structure to be very space efficient since it must store data about a very large dataset. For this structure, space efficiency is more important than accuracy since small deviations in such a large dataset should not be significant because these deviations in past data should not greatly effect what is currently trending.

For the latter data structure, accuracy is more important than space efficiency since the structure contains data which more closely relates to which topics are currently trending and the size of the dataset is much smaller.

4.1.1 Data structure for past tweets

To store data about all occurrences of every hashtag seen in the past, we use a modified version of the time-aggregated Hokusai system[1], which is an extension of the Count-Min sketch. This data structure is described in full detail in section ??.

The goal of the time-aggregated Hokusai system is to store older data at decreased precision since the older data also has decreased value in the computation. The time-aggregated Hokusai system works by storing aggregate data in Count-Min sketches with a 2^i day resolution for the past 2^i days. Each of these Count-Min sketches computes m hash functions from $\{0,1\}^*$ to $\{0,1,,n-1\}$.

To this structure we add another layer that combines these Count-Min sketches into a single Count-Min sketch that depreciates the value of older data. This aggregate Count-Min sketch stores the values $\bar{M} + \sum_{j=1}^{\log(T)} \frac{M^j}{2^j}$ where $\bar{M}$ is the Count-Min sketch storing today's data, M^j is the Count-Min sketch storing the data for the sketch with a 2^j resolution, and T is an upper bound on the number of days stored by the data structure. In section ?? we will show that this aggregate Count-Min sketch weights the hashtags that occurred i days ago with value approximately $\frac{1}{2i}$.

4.1.2 Data structure for current tweets

Our data structure for holding hashtags seen in the last y minutes consists of three components: a max Fibonacci heap, a queue, and a hash table.

The keys for the hash table are the hashtags, and the values stored in the table are (frequency, pointer to corresponding node in heap) pairs.

The queue contains (hashtag, timestamp) pairs, each of which is inserted upon seeing a hashtag in the input stream.

The heap has keys equal to (frequency in last y minutes) / (value in Hokusai data structure) for a specific hashtag, and the value stored in the node is the corresponding hashtag.

4.1.3 Updating Hokusai data structure

Algorithm ?? describes the necessary steps to maintain the Hokusai data structure as new input is provided.

[1]http://arxiv.org/pdf/1210.4891v1.pdf

Algorithm 1 Time Aggregation for the Hokusai structure

for all i **do**
  Initialize Count-Min sketch $M^i = 0$
Initialize $t = 0$, $\bar{M} = 0$, and $A = 0$
while data arrives **do**
  Aggregate data into sketch $\bar{M}$ for current day while also adding this data to A
  $t \leftarrow t + 1$ (increment counter)
  $A \leftarrow A - \bar{M}$
  for $j = 0$ to argmax $\{l \text{ where } t \bmod 2^l = 0\}$ **do**
    $A \leftarrow A + 2^{-j}(\bar{M} - M^j)$
    $T \leftarrow \bar{M}$ (back up temporary storage)
    $\bar{M} \leftarrow \bar{M} + M^j$ (increment cumulative sum)
    $M^j \leftarrow T$ (new value for M^j)
  $\bar{M} \leftarrow 0$ (reset aggregator)

4.1.4 Updating heap and hash tables

Algorithm 2 Thread to process data input.

while data arrives **do**
  if hashtag is in hash table **then**
    Increment the frequency stored at that entry.
    Update the key of the node in the Fibonacci heap.
  else
    Insert a new node into the Fibonacci heap with the appropriate key and value.
    Insert the hashtag into the hash table with a pointer to this node.

4.1.5 Finding the trending hashtags

4.2 Analysis

Maybe put table here of all tunable parameters mentioned in the paper to remind readers what they refer to.

4.2.1 Time analysis

Let s = size of heap = number of distinct hashtags seen in last y minutes.

Time for actions when a hashtag comes in:

Algorithm 3 Thread to discard oldest recent hashtags.

loop
  if queue is not empty **then**
    Peek at end of queue.
    if the timestamp + y minutes is before the current time **then**
      Look up the hashtag in the hash table and decrement the stored frequency.
      if the frequency is now 0 **then**
        Delete the node in the heap pointed to by this entry in the table.
        Delete this entry in the hash table.
      else
        Update the key of this node pointed to by this entry in the table to the proper value given the new frequency.
Make the hash table for the oldest x minute interval correspond to the upcoming x minute interval.

Algorithm 4 Updates at end of a day.

Do all the end-of-day aggregation for the Hokusai structure as detailed in Algorithm **??**.
for all elements in the Fibonacci heap **do**
  look up the hashtag corresponding to this node in the hash table
  Update the key of the node to:

$$\frac{\text{frequency in last y minutes found at the table entry}}{\text{new value in Hokusai data structure}}$$

Algorithm 5 Algorithm to find top k trending items.

Perform k delete-max operations on the Fibonacci heap, storing each of the nodes deleted in a list L.
for all nodes N in L **do**
  Announce that the hashtag associated with N is trending.
  Insert N into the Fibonacci heap.

Takes amortized O(m) time to update Hokusai structure.

Takes expected O(1) time to check if in hash table. If so:

O(1) time to increment

O(m) time to compute key

O(1) amortized time to update key since key nondecreasing

Else:

O(m) time to compute key value and O(1) time to insert node in heap

Takes O(1) time to insert into hash table.

Thus, takes O(m) amortized time + expected O(1) time to do the actions when a hashtag comes in.

Time for actions when y minutes pass after a hashtag comes in: O(1) time to check if queue is empty. If not: O(1) time to look at the queue and see if the timestamp + y minutes is before the current time. If so: Expected O(1) time to look up in hash table and decrement the frequency. O(1) time to check if frequency is 0. If so: O(log(s)) amortized time to delete node in heap. O(1) time to delete entry in hash table. Else: O(m) amortized time to compute the O(log(s)) amortized time to update the heap given this key.

Thus, takes O(log(s)) amortized time + expected O(1) time + O(m) time to do actions when a hashtag is discarded.

Time for actions when day ends: Takes O(nm) amortized time to do all end of day aggregation. Takes O(m) time to compute each key. Takes O(1) amortized time to update each key since all keys are nondecreasing.

Thus, takes O(nm + s) amortized time + O(ms) time at the end of the day.

Time for actions when finding top k trending items: Takes O(k*log(s)) time for delete-max operations. Takes O(k) time to announce these as trending. Takes O(k) amortized time to reinsert these nodes into the heap.

Thus, takes O(k*log(s)) time + O(k) amortized time to determine whats trending.

4.2.2 Space analysis

Hokusai structure takes O(nm*log(T)) space.

Heap takes O(s) space.

Hash table needs O(s) space to keep expected O(1) lookup time.

Queue requires O(t) space. (where t max hashtags seen in y min interval)

Thus, everything requires O(nm*log(T) + s + t) space.

4.2.3 Correctness

This algorithm finds the k hashtags that have the maximum value of (frequency in last y minutes) / (value in Hokusai data structure).

Claim 1. *The value for hashtag x in the aggregate Count-Min sketch of the Hokusai data structure is within a factor of 4 of $\bar{M}(x) + \sum_{i=1}^{T} \frac{1}{i} *$ (value for x in a Count-Min sketch using the same hash functions for all hashtags occurring i days ago).*

Proof. First, we use Theorem 4 of the Hokusai paper[2] which states that "At t, the sketch M^j contains statistics for the period $[t-\delta, t-\delta-2^j]$ where $\delta = t \mod 2^j$."

Let b be the location in the aggregate Count-Min sketch containing the value returned when x is queried.

Let h be any instance of any hashtag that appeared on day m such that seeing h incremented counters in position b.

Case 1: $m = t$

Then the statistics for h are recorded in $\bar{M}$ and are not in any M^j.

Case 2: $2^i > t - m \geq 2^{i-1}$ for some $i > 0$

By Theorem 4, for all $j \leq i-2$, M^j does not contain statistics for m since $m \leq t - 2^{i-1} \leq t - \delta - 2^{i-2}$.

Therefore, the increments that occurred in the Count-Min sketch for hashtags occurring i days ago contribute at most $\sum_{j=i-1}^{T} 2^{-j} < 2^{2-i}$ to the value in position b.

Let k be the largest j such that $t - \delta - 2^j \geq m$

Then $m \leq t - \delta - 2^k$. Let $\lambda = t \mod 2^{k+1}$. Then $\lambda = \delta$ or $\lambda = \delta + 2^k$.

Since k is the largest j such that $t - \delta - 2^j \geq m$, $m > t - \lambda - 2^{k+1}$.

Also, $m \leq t - \delta - 2^k \leq t - \lambda$, so M^{k+1} contains statistics about h.

For all $j \geq i$, $t - \delta - 2^j < t - 2^j < m$, so $k \leq i - 1$.

Thus, incrementing the counter in M^{k+1} contributed at least 2^{-i} to the value in position b.

Thus, the contributions to the sum are within a factor of 4 of $\frac{1}{t-m}$.

Therefore, summing over all hashtags that increment counters in position b gives $\bar{M}(x)$ for all hashtags that occurred on day t, and within a factor of 4 of $\sum_{i=1}^{T} \frac{1}{i} *$(value for x in a Count-Min sketch using the same hash functions for all hashtags occurring $i > 0$ days ago). $\square$

[2] http://arxiv.org/pdf/1210.4891v1.pdf

This value is approximately (frequency in last y minutes) / (freq today $+ \sum_{i=1}^{T} \frac{1}{i} *$ (frequency of hashtag i days ago)). This seems to be a desirable function to maximize since it finds hashtags that are common in the last y minutes that have been comparatively infrequent in the past. This function is good since it especially emphasizes hashtags that are different than those seen in the past few days. This ensures that the same items do not stay trending for too long.

5 Experiments

6 Variants

6.1 Heap function

6.2 Modify heap to always know what's trending

(Good for accuracy, bad since forfeits gain from using Fibonacci heap since items will be deleted from tree more frequently.)

6.3 Deamortizing end of day cost