Article Title

Evan Miller

Kiran Vodrahalli

Albert Lee

January 13, 2015

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1. Introduction

Since its founding in 2006, Twitter has grown into a corporation dominating a significant proporition of social media today. In recent years, it has become interesting to analyze tweets to determine interesting phenomena ranging from the viral spread of news to the detection of earthquakes [**Burks2014**]. The volume of existing tweets is prohibitively large for standard methods of analysis, and requires approaches that are able to either store large amounts of data in memory for fast access (including parallel approaches to data processing) or approaches that only require sublinear memory while retaining accuracy guarantees. In this paper, we investigate a problem utilizing the latter method.

1.1. Problem Statement

We would like to create an online algorithm for providing a real-time estimate of the frequencies of hashtags on Twitter time series data.

why we don't use raw frequency counts for the past hour: there are some hashtags that are constantly present (for example, #porn, #gamesinsight, #teamfollowback (a way to get Twitter followers fast)) these should not be counted as 'trending', so we need to filter these out with the history. what remains is 'what is trending'

Furthermore, our estimate should not require the entire history of tweets

Essentially, we must estimate a probability distribution for a finite set of labels in some moving time window of fixed size. Our labels will be Twitter hashtags, and we will take the window size to be 3 hours. Our aim will be to approximate the k most popular Twitter hashtags in the past 3 hours in order to tell what is trending.

The idea is that estimating the top k hashtags in some time interval provides a model for the topics that are trending on Twitter in that time period.

1.2. Previous Work

Some approaches attempt to do this by storing all of the frequency data, and looking at recent spikes while conditioning on all of the past frequency data in order to determine estimates of trending likelihood. We would like to improve the space complexity of this solution. Our approach will differ in that we will not store all the data of the past, but instead use several Count-Min Sketches to approximate the past frequencies.

MIT people played with Tweet data to predict future trends.

As a preliminary starting point for approximating the past frequencies, we will utilize concepts from 'Hokusai – Sketching Streams in Real Time' [Matusevych2012] to generalize the Count-Min Sketch scheme to time-series data. The rough idea behind this approach is that in the distant past, we should only care about heavy-hitters, i.e. hashtags with high frequencies in order to estimate the likelihood that the hashtag is trending again.

We describe the Hokusai data structure here.

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 each 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 $\{0,1,,n-1\}$.

1.3. Our Approach

We will store the exact counts for the present (3 hour window), and continuously update the past and present as new data streams in.

As a side note, we plan to represent the rolling 3-hour window of frequencies as a discrete approximation: we have a bucket for every second, and update the count of the bucket in every second. When the second passes, we appropriate the bucket that represents the oldest second (i.e. 180 seconds ago) for the newest second, and thus maintain a rolling window across time for exact frequency counts in the past 3 minutes. This rolling window will be denoted as the present.

The baseline comparison for our performance will be the naive version of frequency tallying – we will keep track of the entire history of frequencies, and will use the past to inform the present probability as to whether or not a given hashtag is trending. We also plan to provide a graphic of the top k hashtags, with histogram changing in real time as the estimated frequencies change. Regarding the data, we would ideally gain access to Twitter's firehose of tweets (as only a small subset of the true data is provided for those without access).

2. Describing the Algorithm

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 the deviations in past data should not greatly affect 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.

2.1. Data Structures

2.1.1 History Data Structure

To store data about all occurrences of every hashtag seen in the past, we use a modified version of the time-aggregated Hokusai system [Matusevych2012], which is an extension of the Count-Min sketch. We previously described this data structure in **Previous Work**.

To the Hokusai structure we add another Count-Min sketch that combines the information from the Hokusai Count-Min sketches. We call this external Count-Min sketch the Kernel, since it acts as a weighting function on the CM sketches in the Hokusai structure. Its role is to depreciate the value of older data. Denoting the CM sketches of the Hokusai structure as a vector $\mathbf{M} = \{\overline{M}, M^0, M^1, ..., M^{\lfloor \log(T) \rfloor}\}$, 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. Then, the kernel function is given by

$$k(\mathbf{M}) = \overline{M} + \sum_{j=0}^{\lfloor \log(T) \rfloor} \frac{M^j}{2^j} \tag{1}$$

For each hashtag, the kernel sketch stores a positive value between 0 and 2 (though typically ≤ 1) that approximates how often the hashtag showed up in the past. In **Correctness** we will show that this aggregate Count-Min sketch weights the hashtags that occurred i days ago with weight approximately $\frac{1}{i}$.

2.2. Algorithm Pseudocode

3. Analyzing the Algorithm

3.1. Correctness

3.2. Spatial Analysis

3.3. Runtime Analysis

4. Design Choices

4.1. Choosing the parameters y and z

4.2. Parameters of the History Data Structure

4.3. Choosing the Flavor of Heap

4.4. Choosing the Current-Window Heap Function

5. Results

5.1. Performance Measurements

5.1.1 Naive Algorithm

space measurements (running on one month's worth)

time (running over one month): a few hours

5.1.2 Space-Efficient Algorithm

space measurements (running on one month's worth)

time (running over one month): a few hours

5.2. Are the Algorithm Outputs Actually Trending?

5.2.1 Naive Algorithm

5.2.2 Space-Efficient Algorithm

6. Discussion

do well to get month's worth of data in a few hours (2-3)

implementation wouldnt run in python, do it in C or something

for real data from firehose (all of it), we would use better infrastructure dedicated to processing all the data would use parallel setup (this scheme is adaptable to parallel setup)

seems reasonable if we have proper data storage setup.

7. Future Work

Given enough time, another data stream of interest could be Wikipedia edits – our goal would be to estimate which Wikipedia topics are being edited the most at any given time interval of 3 hours (though we could shrink this to smaller times). As a final sidenote, another application of this algorithm/ data structure would be to estimate the hottest selling stocks on Wall St. Of course this would require a firehose data stream to Wall St., and as that is not as easily obtainable as say, Twitter data, we only mention it as another useful application.

8. Appendix I: Code

We provide links to all our code.

References

[Burks2014] Burks, L., Miller, M., and Zadeh, R. (2014). RAPID ESTIMATE OF GROUND SHAKING INTENSITY BY COMBINING SIMPLE EARTHQUAKE CHARACTERISTICS WITH TWEETS. *Tenth U.S. National Conference on Earthquake Engineering Frontiers of Earthquake Engineering*. Retrieved from http://www.stanford.edu/ rezab/papers/eqtweets.pdf

[Cormode2005] Cormode, G., and Muthukrishnan, S. (2005). An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms, 55*(1), pp. 58-75.

[Matusevych2012] Matusevych, S., Smola, A., and Ahmed, A. (2012). Hokusai–Sketching Streams in Real Time. *UAI '12: 28th conference on Uncertainty in Artificial Intelligence*, pp. 594-603.