The true cost of popularity

The notion of popularity is prevalent within society. We have made charts of the most popular music and movies since the early part of the twentieth century. Elections and referenda are primarily decided by who gets the most votes. Within computer systems, we monitor followers and endorsements in social networks, and track views, hits, and connection attempts in other networks.

Computationally, the problem of determining which items are popular appears at first a straightforward one. Given a dataset of votes, we can simply sort by the item identifier, then count up how many votes are assigned to each. When the number of votes is large, we might try to avoid the overhead of sorting, and aim to more directly pick out the most popular items with only a few passes through the data.

Things get more interesting when we further refine the problem. What happens when the number of votes, and the number of candidate items, gets so large that it is not feasible to keep a tally for each candidate? This might be implausible in the context of political elections, but is an everyday reality in social systems handling many billions of actions (representing votes) on pieces of content or links (representing the items). Here, we may only get one opportunity to see each vote, and must update our data structures accordingly before moving on to the next observation. Other twists complicate things further: what if votes can have different weights, reflecting the intensity of the endorsement? What if these weights can be negative, encoding a removal of support for an item? What if the formula to compute the overall score is not the sum of the weights, but the square of the sum of the weights?

Each of these variations makes the problem more challenging, while only increasing the generality of any solution: if we can create an algorithm to handle all these variations, then it will still work when they do not apply. Such has been the level of interest in designing effective and efficient algorithms that a lexicon has emerged to describe them: the most popular items are the *heavy hitters*; processing each update once as it arrives gives the *streaming model*; allowing negative weights is the (general) *turnstile* model; setting a threshold for being a heavy hitter based on removing the *k* heaviest items is the *k-tail* version; and a weighting function based on squared sums is called l_2. So while the following paper by Larsen et al. addresses the k-tail l_2 heavy hitters problem in the turnstile streaming model, it should be understood as solving a most general version of the problem!

Solutions for more restricted versions of this problem have been defined over the years, and have been put to use in deployments handling large volumes of data. For example, Twitter has used heavy hitter algorithms to track the number of views of individual tweets as they are embedded in different pages around the web[1]. Apple has combined heavy hitter algorithms with privacy tools to allow privately tracking the emerging popularity of words, phrases and emoticons among their users[2]. Broadly speaking, heavy hitter algorithms are defined by two phases: a collection phase to gather data and statistics from viewing the stream of updates, and a search process to extract the heavy hitter items. There are simple and effective randomized algorithms that can create summaries which allow the estimation of the final weight of a given item to a high degree of

[1] https://skillsmatter.com/skillscasts/6844-count-min-sketch-in-real-data-applications

[2] https://machinelearning.apple.com/2017/12/06/learning-with-privacy-at-scale.html

accuracy. However, when there are a very large number of possible items to consider (say, the combination of every tweet and every page on the web), making the search process efficient becomes the chief objective.

Consequently, the main focus of this paper is on building up sufficient information to allow a more effective search process. It proceeds by incrementally developing the solution from first principles, relying on concepts from across computer science: randomly partitioning the input space to simplify the core problem; modifying the encoding of the item identifiers, and applying ideas from coding theory to correct for noise; using a construction based on expander graphs to make this more robust; and finally making use of an approach to clustering from spectral graph theory to ensure that the identifiers of the heavy hitters can be correctly extracted. The end result is an algorithm which for the first time meets the minimum space cost to solve the problem while giving an efficient search time cost.

This opens the way for further work. How efficiently could this clustering approach be implemented in practice, and what applications might it find elsewhere? While identifying popular items is a foundational question for data analysis, there are many more questions that can be asked. The area of streaming algorithms concerns itself with finding efficient algorithms for statistics and queries on large data viewed as a stream of updates. Current challenges revolve around processing massive data sets to extract statistical models for prediction and inference.

Graham Cormode, January 2019