CS 677: Big Data

Count Min Sketch

Lecture 18

Element Frequencies

- Thinking back to our “Majority Sketch”, we discussed several flaws in the algorithm
  - A big one: we generally look for more nuance than just “has more than 50% of occurrences”
  - If things are fairly uniform this information is not helpful
- It would be more helpful to estimate the *frequencies* of each event
  - For instance, the frequency of votes for each candidate

An Algorithm Blooms

- The Frequent Elements Sketch is effective at determining the top N heavy hitters, but let's assume we need frequencies for ALL events
- Time to revisit our old friend, Bloom Filters!
  - Well… counting bloom filters
- In the count-min sketch (CM sketch), build a 2D array of w columns and d rows
- Each row d is associated with a hash function, and column w contains a counter

Count Min Sketch

- As items are inserted, they are hashed once for each row in the 2D array
- Next, column counters are incremented
  - $hash \% w$ = index of the column to increment
- …and, finally, the overall count of items is incremented
- To query the frequency of any event, simply hash it and take the minimum count you find across all the rows
  - (count **min**!!!)

Count Min Intuition

- Why the minimum value? Well, as we know, bloom filters are susceptible to collisions
- By choosing the smallest value we know we will not overestimate the frequency
  - (possibly by a large margin if we've had many collisions in a particular column)
- In fact, our true frequency will always be smaller or equal to our estimate, with predictable errors
  - (just like with regular Bloom filters)

Merging the Sketch

- It is possible to merge distributed CM sketches
- Assumes same array dimensions
- Simply update the counts from each CM instance
- Very useful for parallel/distributed applications (say, maintaining frequencies across several Spark Workers…)

Demo

Let’s check out a demo!

https://florian.github.io/count-min-sketch/