Data Summarization

Trading accuracy for memory and runtime

Yonathan Fisseha

July 1, 2019

Motivation

Typically data is

- Relational
- Limited in volume and speed
  - Carefully generated by a limited number of users

The data we deal with has changed significantly in recent years

- Less structure
- More volume
- More speed
- **Temporal**

Examples

- Telemetry data: Web applications that collect data on user interaction (e.g. clicks)
- Distributed Systems: Clusters that report data on status of nodes (e.g. microservices)
- Sensors: Measurements of the environment reported continuously (e.g. temperature)
- Videos and audio streams: Stream of content from multiple devices (e.g. traffic cameras)

Implications

Relational databases were not designed for this

- No clear relationship in the data –> hard to model
- Not optimized for high ingestion rate (thousands of GB/sec)
- Query time grows quickly with the data size
- Space requirements are usually impossible to meet (PBs or many TBs)
- No built-in abstraction for temporal navigation of data

Moving forward

We can take advantage of some characteristics of timeseries data. We usually:

1. don't care as much about old data
2. need a general understanding of data, not details
3. have append-only needs

Case Study: Netflix

Generating recommendations based on movies people click on...
Insight:

> *A movie someone clicked on years ago is not a useful indicator*

- Compress older data more aggressively (**lossy**) [1]
  - No noticeable performance degradation
- Provide more general statistical summaries for older data
  - Okay with relatively higher error rates

Results

- Saved a *lot* of space (and reduced computation time)
- Improved system design

[1] Not in the typical sense of lossy compression. They drop some columns that aren't useful at that point.

Challenges

Space (and time!) complexity vs accuracy:

- Statistical summaries of interest: aggregates
  - E.g. average, frequency count, median, etc.
  - Trivial solution: store all of the data and run statistical summaries when needed
  - Problem: expensive space usage
- The problem reduces to parameterized lossy compression
  - How lossy do we have to be to get an error bound of x on the data summary?
  - Otherwise, best lossy compression is to simply drop all data

Other Challenges

Data points can arrive out of order and at different rates

Summarizing data

Insight: we need fast insert and incremental update of results

Some methods for approximating statistical summaries within some ϵ error of the raw data

1. **Sampling**
   Estimate the stream based on a subset of the data
2. **Sketches**
   Probabilistic data structures
3. **Transforms**
   Convert the underlying signal to a different domain (e.g. from time to frequency)

Sampling

Naive example:

Let $S = s_1, s_2, ..., s_n$ be a stream where we are processing the n_i element.

Then we can uniformly sample only k elements by using the Reservoir algorithm:

- Keep the first k elements
- Then for the n_i element, we keep it with probability k/i
  - Randomly replace one of the k elements already kept if we keep it
- Now compute statistical queries on the size-k sample instead of S

Sampling Cont'd

Pros

- Simple!
- Space doesn't grow with the size of S
- *insert* is constant time
- Query time is independent of the size of S
- Can incrementally update results

Cons

- Strong assumptions on the distribution of S
- Doesn't take advantage of recent vs old data
- Prone to miss important data points (e.g. heavy hitters)

Improving Sampling

- Main problem: assumption of uniform distribution!
- Idea: use an oracle (or something approximating an oracle) to adjust our assumption
  - Find the frequency of items and avoid over/under sampling
- This is challenging because finding the exact frequencies has a $\Omega(|S|)$ bound
- We must approximate it efficiently
  - These approximation techniques turn out to be more useful that just approximating the frequency

Sketches

- General idea: Consider some data D and some function f, then compress D into C such that we can compute or approximate the function f only using C
- Intuition for our case: we must compress D such that $C << D$ and approximate f using C
- Generally formulated using a probabilistic data structure
- Can be lossy but will use less space and answer queries approximately
- Example, bloom filters
  - Answer is: No or maybe (small probability of false positives)
  - sublinear space
- Often make use of hash functions

Count-min

- Applying the same probabilistic principle towards frequency counting:
- A matrix of size wXd (parameterized by δ and ϵ) and H_d hash functions that are pairwise independent...
- Assume every $s_i \in S$ is a tuple of the form (i_t, c_t) which means we update the t^{th} as $a(t) = a(t-1) + c_t$
- Update the data structure as[2]:
  - Map a function in H to every row of the matrix
  - For each row, insert $a(t)$ into matrix(row, $h_i(i)$)

[2] If items are not always increasing, the solutions need to be generalized slightly more

Count-min: Answering Queries

- Answering queries using count-min:
- Point query: item at i of the stream a:
  - $= min(matrix[row, h(i)])$
- Inner product of axb (both with a count-min data structure):
  - let $(axb)'$ be $\sum count_a[j, k] x count_b[j, k]$
  - then we approximate by doing $min(axb)'$
- Can also support range queries efficiently using dyadic ranges but slightly more complicated
  - maintain a sketch for each set of dyadic ranges of length 2^y
  - for a given query, break it down into $2\log_2(n)$ dyadic ranges to cover the whole range
  - query each of the sketches using point query
  - return the sum of the results

Transforms

- Based on signal processing
- A given vector can be represented by it's transform coefficients X and basis vectors B as $\sum X[i]B[i]$
- We want to approximate the vector using a subset of the basis vectors and the same equations above
- The optimal approximation minimizes the square error between the approximation and the true value
- Instead of transforming it directly, minimize the error square instead
  - *Can be done online* – but less efficient than transform algorithms for the static case
- We can truncate the transformation matrix coefficients (by dropping the coefficients with low energy)
- Apply minimization techniques from here

Other Methods

- Another method to manage streaming timeseries data is lossless compression
- Sufficiently compress the data to fit in the working set memory of the system
- Other storage systems can be used to archive it after summarization
- Problem: overhead of compression *and* decompression
- Possible solution: Algorithm aware lossless compression schemes
  - Compress the data
  - Run statistical summaries
  - Archive the compressed data
  - Directly run on compressed data to satisfy user queries
- The compression scheme must be designed with the algorithms in mind
  - E.g. Succinct

Thanks!