Summary Structures for Massive Data

Graham Cormode

G.Cormode@warwick.ac.uk
“Big” data arises in many forms:
- Physical Measurements: from science (physics, astronomy)
- Medical data: genetic measurements, detailed time series
- Activity data: GPS location, social network activity
- Business data: customer behavior tracking at fine detail

Common themes:
- Data is large, and growing
- There are important patterns and trends in the data
- We don’t fully know how to find them
Making sense of Big Data

♦ Want to be able to interrogate data in different use-cases:
  – **Routine Reporting**: standard set of queries to run
  – **Analysis**: ad hoc querying to answer ‘data science’ questions
  – **Monitoring**: identify when current behavior differs from old
  – **Mining**: extract new knowledge and patterns from data

♦ In all cases, need to answer certain basic questions quickly:
  – Describe the distribution of particular attributes in the data
  – How many (distinct) $X$ were seen?
  – How many $X < Y$ were seen?
  – Give some representative examples of items in the data
Summary Structures

- Much work on building a **summary** to (approximately) answer such questions
- To earn the name, should be (very) small!
  - Can keep in fast storage
- Should be able to build, update and query efficiently
- Key methods for summaries:
  - **Create** an empty summary
  - **Update** with one new tuple: streaming processing
  - **Merge** summaries together: distributed processing
  - **Query**: may tolerate some approximation
Techniques in Summaries

♦ Several broad classes of techniques generate summaries:
  – Sketch techniques: linear projections
  – Sampling techniques: (complex) random selection
  – Other special-purpose techniques
♦ In each class, will outline ‘classic’ and ‘recent’ results
♦ Conclude with “state of the union” of summaries
Random Sampling

- **Basic idea**: draw random sample, answer query on sample (and scale up if needed)
- **Update**: include new item in sample with probability \( \frac{1}{n} \) (and kick out an old item if sample is full)
- **Merge**: draw items from each input sample with the probability proportional to relative input size
- **Query**: run query on the sample (and possibly rescale result)

- **Accuracy**: answers any “predicate query” with additive error
  - E.g. What fraction of input items satisfy property \( X \)?
  - Error +/- \( \varepsilon \) with 95% probability for sample size \( O(1/\varepsilon^2) \)
Structure-aware Sampling

♦ Most queries are actually range queries:
  – “How much traffic from region X to region Y at 2am to 4am?”
♦ Much structure in data [Cohen, C, Duffield 11]
  – Order (e.g. ordered timestamps, durations etc.)
  – Hierarchy (e.g. geographic and network hierarchies)
  – (Multidimensional) products of structures
♦ Make sampling structure-aware when ejecting keys
  – Carefully pick subset of keys to subsample from
  – Empirically: constant factor improvement from same size sample
Sampling Pros and Cons

- Samples are very general, but have some limitations
- Uniform samples are no good for many problems
  - Anything to do with number of distinct items
- For some queries, other summaries have better performance
  - Technically: $O(1/\varepsilon^2)$ vs $O(1/\varepsilon)$ size
  - Practically: may be factors of 10s or 100s
Sketch Summaries

- Subclass of summaries that are linear transforms of input
  - **Merge** = sum
  - Easy to extend to inputs that have negative weights

- Efficient sketches approximate quantities of interest:
  - $O(\varepsilon^{-1})$ space for point queries with $\varepsilon L_1$ error [CM]
  - $O(\varepsilon^{-2})$ space for point queries with $\varepsilon L_2$ error [CCFC]
  - $O(\varepsilon^{-2})$ space to estimate $L_2$ with $\varepsilon$ relative error [AMS]
Count-Min Sketch [C, Muthukrishnan ’03]

- Simple sketch idea, used in many different tasks
- Applicable when input data modeled as vector $x$ of dimension $m$
- Creates a small summary as an array of $w \times d$ in size
- Use $d$ (simple) hash function to map vector entries to $[1..w]$
- (Implicit) linear transform of input vector, so flexible

Array: $CM[i,j]$
Count-Min Sketch Operations

- **Update**: each entry in vector $x$ is mapped to one bucket per row
- **Merge**: combine two sketches by entry-wise summation
- **Query**: Estimate $x[j]$ by taking $\min_k CM[k,h_k(j)]$
  - Guarantees error less than $\varepsilon N$ in size $O(1/\varepsilon \log 1/\delta)$ (Markov ineq)
  - Probability of more error is less than $1-\delta$

- $w = 2/\varepsilon$
- $d = \log 1/\delta$
- $j, +c$
Lp Sampling

- **Lp sampling**: use sketches to sample i w/prob \((1 \pm \varepsilon) \frac{f_i^p}{|f|^p}\)
- “Efficient” solutions developed of size \(O(\varepsilon^{-2} \log^2 n)\)
  - [Monemizadeh, Woodruff 10] [Jowhari, Saglam, Tardos 11]
- Enable novel “graph sketching” techniques
  - Sketches for connectivity, sparsifiers [Ahn, Guha, McGregor 12]

- **Challenge**: improve space efficiency of L_p sampling
  - Empirically or analytically
Sketching Pros and Cons

- “Linear” summaries: can add, subtract, scale easily
  - Useful for forecasting models, large feature vectors in ML
- Other sketches have been designed for:
  - Count-distinct, Set sizes (Flajolet-Martin and beyond)
  - Set membership (Bloom Filter)
  - Vector operations: Euclidean norm, cosine similarity
- Some sketch types are large, slow to update (but parallel)
- Tricky to adapt to large domains (e.g. strings)
- Don’t support complex operations (e.g. arbitrary queries)
Special-purpose Summaries

- **Misra-Gries (MG)** algorithm finds up to \( k \) items that occur more than \( 1/k \) fraction of the time in the input
- **Update**: Keep \( k \) different candidates in hand. For each item:
  - If item is monitored, increase its counter
  - Else, if < \( k \) items monitored, add new item with count 1
  - Else, decrease all counts by 1
Streaming MG analysis

♦ N = total weight of input
♦ M = sum of counters in data structure
♦ **Error** in any estimated count at most \((N-M)/(k+1)\)
  – Estimated count a lower bound on true count
  – Each decrement spread over \((k+1)\) items: 1 new one and \(k\) in MG
  – Equivalent to deleting \((k+1)\) distinct items from stream
  – At most \((N-M)/(k+1)\) decrement operations
  – Hence, can have “deleted” \((N-M)/(k+1)\) copies of any item
  – So estimated counts have at most this much error
Merge algorithm:
- Merge the counter sets in the obvious way
- Take the \((k+1)\)th largest counter \(= C_{k+1}\), and subtract from all
- Delete non-positive counters
- Sum of remaining counters is \(M_{12}\)

This keeps the same guarantee as Update:
- Merge subtracts at least \((k+1)C_{k+1}\) from counter sums
- So \((k+1)C_{k+1} \leq (M_1 + M_2 - M_{12})\)
- By induction, error is
  \[
  ((N_1 - M_1) + (N_2 - M_2) + (M_1 + M_2 - M_{12}))/(k+1) = ((N_1 + N_2) - M_{12})/(k+1)
  \]
  (prior error) (from merge) (as claimed)
Special Purpose Summaries: Pros and Cons

- Tend to work very well for their target domain
- But only work for certain problems, not general
- Other special purpose summaries for:
  - Summarize distributions (medians): q-digest, GK summary
  - Graph distances, connectivity: limited results so far
  - (Multidimensional) geometric data: for clustering, range queries
    - Coresets, $\varepsilon$-approximations, $\varepsilon$-kernels, $\varepsilon$-nets
Applications shown for Summaries

- Machine learning over huge numbers of features
- Data mining: scalable anomaly/outlier detection
- Database query planning
- Password quality checking [HSM 10]
- Large linear algebra computations
- Cluster computations (MapReduce)
- Distributed Continuous Monitoring
- Privacy preserving computations
- ... [Your application here?]

More speculative
Summary of Summary Issues

Strengths

♦ (Often) easy to code and use
  – Can be easier than exact algs

♦ Small — cache-friendly
  – So can be very fast

♦ Open source implementations
  – (maybe barebones, rigid)

♦ Easily teachable
  – As intro to probabilistic analysis

♦ (Mostly) highly parallel

Weaknesses

♦ (Still) resistance to random, approx algs
  – Less so for Bloom filter, hashes

♦ Memory/disk is cheap
  – So can do it the slow way

♦ Not yet in standard libraries
  – Developing: MadLib, Stream-lib

♦ Not yet in courses / textbooks
  – “this CM sketch sounds like the bomb! (although I have not heard of it before)”

♦ Few public success stories
Resources

- Sample implementations on web
  - Ad hoc, of varying quality
- Technical descriptions
  - Original papers
  - Surveys, comparisons
- (Partial) wikis and book chapters
  - Wiki: sites.google.com/site/countminsketch/
  - “Sketch Techniques for Approximate Query Processing”
    dimacs.rutgers.edu/~graham/pubs/papers/sk.pdf
Example: Bloom Filters (1970)

♦ A well-known and widely used summary
♦ Bloom filters compactly encode set membership
  – **Create**: Pick *k* hash functions to map items to (empty) bit vector
  – **Update**: Hash and set *k* entries to 1 to indicate item is present
  – **Merge**: Take bit-wise OR of two Bloom Filter vectors
  – **Query**: Hash item to vector, assume in set if all *k* entries are 1
♦ **Analysis**: store set size *n* in \( \sim 10n \) bits with few false positive