Final Project Meeting Signups Out  
Quiz 2 Review Monday  
Quiz 2 Wednesday  

We've left off the slides from CMT from the deck; it won't be covered on the exam.
Sampling and Sketching
Do We Always Need to Process All the Data?

• For many data analytics applications, it may not be necessary to look at every record.

• E.g., suppose we want to see how revenue changed over the past 12 months

  • Could scan all data

  or

  • Could randomly sample data and compute estimate / error bars
Error Bars: Central Limit Theorem

• Given a population with a finite mean $\mu$ and a finite non-zero variance $\sigma^2$, the sampling distribution of the mean approaches a normal distribution with a mean of $\mu$ and a variance of $\sigma^2/N$ as $N$, the sample size, increases.

• Here, the sampling distribution of the mean is the distribution of the means of samples of the dataset

• This means we can estimate the mean, and estimate the error in the mean
  • $\mu = \text{mean}(\text{sample})$
  • $\sigma = \frac{\text{stddev}(\text{sample})}{\sqrt{N}}$, $\text{stddev}(\text{sample}) = \sqrt{\frac{\sum_{i \in \text{sample}} (i - \mu)^2}{N}}$

*Similar closed form solutions for sum, count, and other simple statistics*
What if CLT Doesn’t Apply

• E.g., suppose you want error bars on the median, or on percentiles in a histogram

• Or some complex predictive function, e.g., some ML algorithm

• The Nonparametric Bootstrap is a generic technique for this
  • Idea: repeatedly resample a sample
Bootstrap Method

Given a function $F$ and a sample $S$ of size $N$, with parameter $K$ (the number of bootstraps)

Goal is $\pm p$ confidence interval

For $i$ in 1 .. $K$
  - $S_{\text{new}} = \text{sample of size } N \text{ of } S \text{ with replacement}$
  - $\text{Results}[i] = F(S_{\text{new}})$

Sort results, return $p$, $1-p$ percentile of results
Example

Data:
[36, 23, 7, 25, 27, 31, 27, 10, 11, 8, 21, 4, 41, 0, 20, 5, 0, 36, 40, 10, 12, 31, 24, 2, 28, 8, 9, 25, 48, 43, 40, 2, 26, 0, 2, 5, 32, 9, 0, 10, 33, 1, 23, 7, 39, 18, 32, 16, 40, 4, 42, 28, 28, 26, 42, 0, 45, 25, 10, 13, 31, 3, 11, 28, 25, 23, 16, 31, 2, 6, 34, 19, 48, 27, 48, 39, 40, 6, 3, 28, 26, 19, 34, 38, 42, 1, 47, 22, 7, 36, 38, 35, 35, 42, 49, 41, 40, 11, 10, 1, 1]

Sample:
[25, 10, 35, 25, 23, 0, 20, 24, 23, 25, 6, 42, 40, 38, 40, 4, 8, 16, 38, 8]

Resample 1: [42, 40, 8, 25, 0, 42, 24, 0, 16, 42, 23, 25, 25, 10, 40]  Mean = 24.1
Resample 2: [23, 25, 10, 42, 23, 0, 0, 24, 23, 23, 38, 25, 16, 35, 25]  Mean = 22.1
Resample 3: [6, 38, 40, 23, 23, 40, 23, 4, 8, 25, 4, 8, 25, 20, 0]  Mean = 19.13

Mean = 22.91
Mean = 22.1
Mean = 19.13
Resulting Means after 100 runs


Confidence interval of mean 16.6 ... 28.87
Why Does This Work

• A random sample is an approximation of the distribution of the data
  • If it’s big enough, it’s a good approximation

• Resampling the sample is close to resampling from the original data
  • Variation in those samples captures variation in the original data
  • Of course, it will miss outliers, extrema, etc.
  • But it will work well for a variety of descriptive statistics, including quantiles, regression errors, precision/recall estimates, etc.
When Doesn’t This Work

• Your sample needs to be big enough (N > 20 is a rule of thumb, but it will vary a lot depending on data)
• It won’t work for extrema (e.g., min / max)
• It won’t work well for highly structured data (i.e., you can’t randomly sample a graph, compute the average connectivity, and expect to get something meaningful)
• It won’t work if your sample is not truly random
Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, Ion Stoica. **BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data.** *In ACM EuroSys 2013*
Ultimate Goal of BlinkDB

• **Observation:** Many applications can tolerate quick, approximate answers over data

• **Trade-off:** few percent error for up orders of magnitude in efficiency

• Acceptable in decision support, recommendation system, diagnosis, root cause analysis
Overview

• **Problem**
Users are overwhelmed by data volumes AND increasingly want to compute sophisticated statistics over their data. Existing database systems do not satisfy their needs.

• **Our Goal**
Provide interactive ad-hoc analytical (SQL) queries over very large data sets.

• **Basic Approach**
Run queries over stored/precomputed samples, providing answers with bounded errors for arbitrary functions.
Challenges/Solutions

**Generality:** Accurate error estimates for complex SQL statements and user-defined functions
  - Investigating techniques like bootstrap and jack knife for providing error estimates for arbitrary user-defined (differentiable) functions

**Flexibility/Reliability:** Accurate estimations of response times for ad hoc queries (including over small domains)
  - Using stratified sampling rather than random sampling

**Parallelism/Scalability:** Sub-second latencies for parallel queries running on hundreds of machines
  - Not doing online aggregation, but pre-computing samples
  - Optimization problem!
System Architecture

Original Data
System Architecture

Offline-sampling: multiple data samples at various granularities and across different dimensions (columns)
Initial Prototype

Samples striped over 100s or 1,000s of machines both on disks and in-memory (i.e., RDDs)
System Architecture

Sample Selection

HiveQL/SQL Query

SELECT foo (*) FROM TABLE;

Query Plan

Sampling Module

Original Data

On-Disk Samples

In-Memory Samples

Predict cost and error for ad-hoc queries using smaller samples and historical context
System Architecture

HiveQL/SQL Query

Original Data

Query Plan

Sample Selection

Online sample selection to pick best sample(s) based on query latency and accuracy requirements
System Architecture

SELECT * FROM TABLE;

HiveQL/SQL Query

Query Plan

Sample Selection

Hive

Error Bars & Confidence Intervals

Result

182.23 ± 5.56
(95% confidence)

Parallel query execution on multiple samples striped across multiple machines

Original Data

Sampling Module

On-Disk Samples

In-Memory Samples

In-Memory

Parallel query execution on multiple samples striped across multiple machines
System Architecture

HiveQL/SQL Query

TABLE

Original Data

Sampling Module

On-Disk Samples

In-Memory Samples

Query Plan

Sample Selection

Error Bars & Confidence Intervals

Hive

Result

182.23 ± 5.56 (95% confidence)

Error Bars & Confidence intervals using bootstrap
Handling Rare Values

• Some values in tables much less popular

Q1: SELECT avg(Salary) FROM employees WHERE city='New York'
Q2: SELECT avg(Salary) FROM employees WHERE city='Cambridge'

Solution: Stratified sampling – only sample values that appear more than K times; preserve other values
Example

TABLE

Sess. Genre OS City URL

Query Templates

<table>
<thead>
<tr>
<th>Field</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>City</td>
<td>30%</td>
</tr>
<tr>
<td>Genre</td>
<td>25%</td>
</tr>
<tr>
<td>Genre AND City</td>
<td>18%</td>
</tr>
<tr>
<td>URL</td>
<td>15%</td>
</tr>
<tr>
<td>OS AND URL</td>
<td>12%</td>
</tr>
</tbody>
</table>

Family of random samples

Family of stratified samples on {City}

Family of stratified samples on {OS, URL}
What Samples to Create

1. Always maintain a uniform sample
2. For stratified samples, start from past “query templates”
3. Choose the combinations of columns that are “best” for those templates
   • Favor Non-uniform columns
4. Avoid “over-fitting” the past workload
   • Favor sample families useful for answering queries not captured by exiting templates
Experimental Setup

• 30-day log of media accesses by users from a video analytics company. Raw data 17 TB, partitioned this data across 100 nodes.

• Log of 20,000 queries (a sample of 200 queries had 42 templates).
Results

<table>
<thead>
<tr>
<th></th>
<th>2.5 TB</th>
<th>7.5 TB</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlinkDB 1% error</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Hive</td>
<td>2000</td>
<td>6000</td>
</tr>
</tbody>
</table>

Runtime Vs. Dataset Size

Runtime (s, log scale)

- 2.5 TB
- 7.5 TB
BlinkDB – Summary

• A massively parallel DB that supports ad-hoc queries with error and response-time bounds.
• An optimal strategy for building & maintaining multi-dimensional, multi-granularity samples
• Dynamic Query Cost Estimation and Sample Selection
Extreme Statistics

• What about cases where you need to estimate the max, min, # of distinct values etc?

• Sampling won’t work

• No free lunch: Need to look at all of the values

• For min/max, can keep a running value

• But what about distinct values, top-N, etc?
Sketching Algorithms

Count distinct: hyperloglog
Heavy hitters (top K): countmin
Quantiles (median): quantile sketch

...
How many samples on average until there are $k$ trailing zeros?

<table>
<thead>
<tr>
<th>Sample</th>
<th>Binary</th>
<th>Clicker:</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>0b110010</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0b101000</td>
<td>a. $k$</td>
</tr>
<tr>
<td>35</td>
<td>0b100011</td>
<td>b. 1</td>
</tr>
<tr>
<td>25</td>
<td>0b110010</td>
<td>c. $2^k$</td>
</tr>
<tr>
<td>23</td>
<td>0b101110</td>
<td>d. $k^2$</td>
</tr>
<tr>
<td>0</td>
<td>0b000000</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0b101000</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>0b110000</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>0b101110</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>0b110010</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0b110000</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>0b101010</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>0b101000</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>0b100110</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>0b101000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0b100000</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0b100000</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0b100000</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>0b100110</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0b100000</td>
<td></td>
</tr>
</tbody>
</table>
How many samples on average until there are $k$ trailing zeros?

<table>
<thead>
<tr>
<th>Sample</th>
<th>Binary Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>0b110010</td>
</tr>
<tr>
<td>10</td>
<td>0b101000</td>
</tr>
<tr>
<td>35</td>
<td>0b100011</td>
</tr>
<tr>
<td>25</td>
<td>0b110010</td>
</tr>
<tr>
<td>23</td>
<td>0b101110</td>
</tr>
<tr>
<td>0</td>
<td>0b000000</td>
</tr>
<tr>
<td>20</td>
<td>0b101000</td>
</tr>
<tr>
<td>24</td>
<td>0b110000</td>
</tr>
<tr>
<td>23</td>
<td>0b101110</td>
</tr>
<tr>
<td>25</td>
<td>0b110010</td>
</tr>
<tr>
<td>6</td>
<td>0b110000</td>
</tr>
<tr>
<td>42</td>
<td>0b101010</td>
</tr>
<tr>
<td>40</td>
<td>0b101000</td>
</tr>
<tr>
<td>38</td>
<td>0b100110</td>
</tr>
<tr>
<td>40</td>
<td>0b101000</td>
</tr>
<tr>
<td>4</td>
<td>0b100000</td>
</tr>
<tr>
<td>8</td>
<td>0b100000</td>
</tr>
<tr>
<td>16</td>
<td>0b100000</td>
</tr>
<tr>
<td>38</td>
<td>0b100110</td>
</tr>
<tr>
<td>8</td>
<td>0b100000</td>
</tr>
</tbody>
</table>

Clicker:

- a. $k$
- b. $1$
- c. $2^k$
- d. $k^2$
Hyperloglog Algorithm – Approach 0

Given a vector of values, \( V \), compute \( H(v) \) for all \( v \) in \( V \)

- \( H \) is a hash function that goes from \( v \) to a large random integer

MaxZeros = 0

For each \( h \) in \( H(v) \):

- Zeros = count the number of leading zeros in \( h \)
- MaxZeros = max(Zeros, MaxZeros)

Distinct \( \text{vals} \) = \( 2^{\text{MaxZeros}} \)
Discussion

• This is an accurate estimator, but it is noisy
• We can do better by averaging a bunch of estimators

• Could repeat the previous algorithm N times, but requires computing N hashes per data item, which is expensive

• This is the problem hyperloglog tries to solve
Hyperloglog Algorithm – Approach 1

Idea: split hash value into m “bucket” bits and 128 – m “value” bits; store 2^m max’s

<table>
<thead>
<tr>
<th>m “bucket” bits</th>
<th>128 – m hash bits</th>
</tr>
</thead>
</table>

Given a vector of values, V, compute H(v) for all v in V

H is a hash function that goes from v to a large random integer

MaxZeros = [0, 0, ..] // length 2^m

For each h in H(v):

- bucket = bits 0 … m-1 of h
- value = bits m … 128 of h
- zeros = count the number of leading zeros in value
- MaxZeros[bucket] = max(zeros, MaxZeros[bucket])

Distinct vals = \( \text{avg}(2^{\text{MaxZeros}[0]}, \ldots, 2^{\text{MaxZeros}[2^m]}) \)

Creates 2^m hashes out of a single hash
Algorithm 1 Discussion

• Paper shows that taking the harmonic mean of the estimates, instead of the average, results in a better estimate. \[ H(1,3,4) = \left( \frac{1^{-1} + 4^{-1} + 4^{-1}}{3} \right)^{-1} = \frac{3}{\frac{1}{1} + \frac{1}{4} + \frac{1}{4}} = \frac{3}{1.5} = 2. \]

• Error is \(1.04/\sqrt{m}\), where \(m\) is the number of maximums we maintain

• Discarding outlier buckets also helps

• Also can be updated – i.e., merged with another set of counters to get a new estimate of the cardinality
Summary