A Review of Data Canopy

Keith Daniel Lovett

Data Exploration

- Statistics:
  - Reveal trends in data.
  - Act as building blocks for machine learning formulas. (Which can help reveal more nuanced trends we may not otherwise notice.)
- We want a means of exploring data using statistics that is both fast and flexible.

What’s Flexible? What’s Fast?

FLEXIBLE (lots of functionality for statistics, no data management) -
Numpy
Modeltools

FAST (limited statistical functionality, good data management) –
Database Systems

Database Connectors:
Psycopg,
MonettDB.R,
SciDB-Py

Can we make a system for data exploration which…

FLEXIBLE (lots of functionality for statistics, no data management) -
Numpy
Modeltools

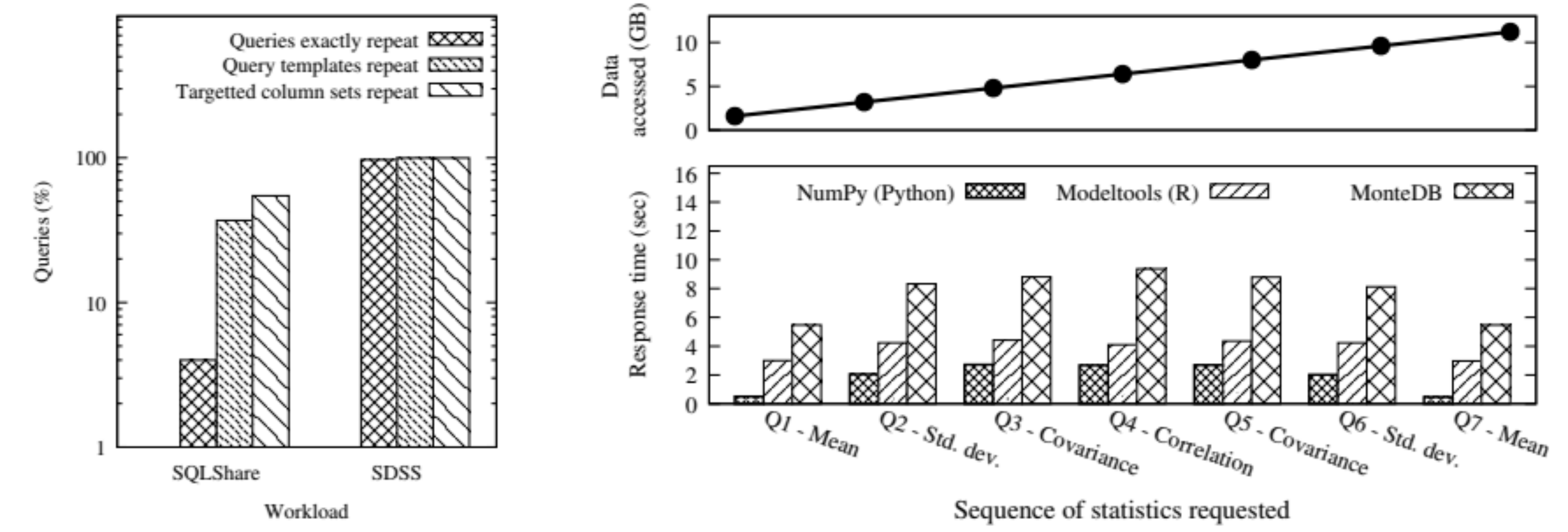
FAST (limited statistical functionality, good data management) –
Database Systems

Database Connectors:
Psycopg,
MonettDB.R,
SciDB-Py

Maintains the flexibility offered by software libraries

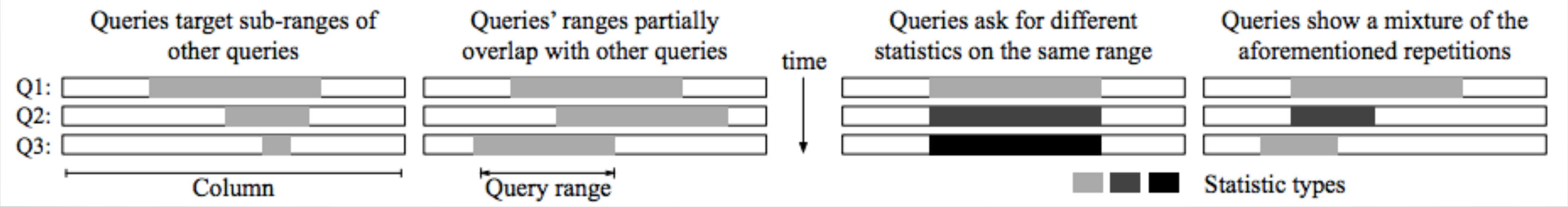
Offers speeds faster than those found when using database connectors

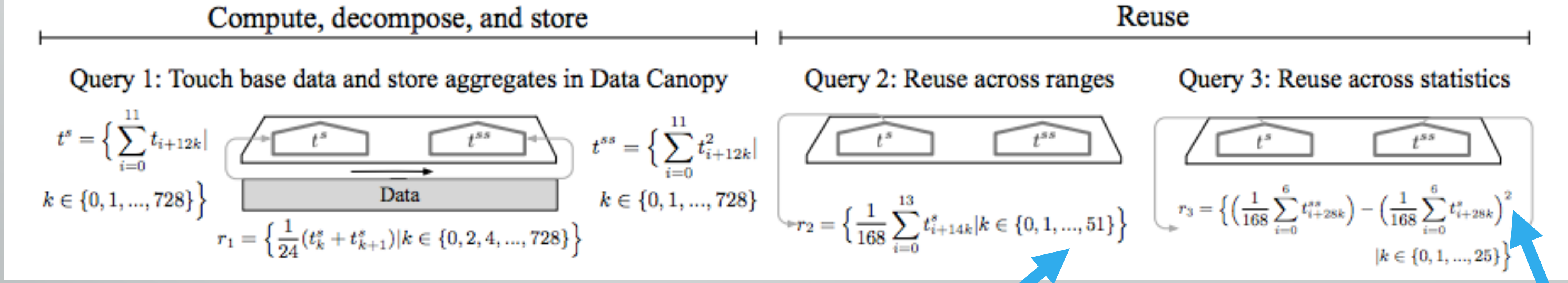
What’s the hold-up?



…Data reusability!

How do we Define Query "Similarity"





Sub-range

Sub-range, different statistic

Great!

- We've identified our problem (data-reusability) and defined the types of ways in which data is potentially reusable…
- … But how do we reuse it?

?

The Solution? Aggregates!

- We can form a smart cache (a “Data Canopy”) of basic aggregates of data! We can think about these aggregates in two ways:
  - Those immediately needed
  - Those not yet needed, but which can be formed from those which were immediately needed.
- Example:
  - Query 1: Request for mean temps for each day.
  - Query 2: Request for mean temps for each week. (different granularity from first query.)
  - Query 3: Request for variances in temps for every two weeks.

$$v_x = \left(\frac{1}{N}\sum_{i=1}^{N} x_i^2\right) - \left(\frac{1}{N}\sum_{i=1}^{N} x_i\right)^2$$

A More f(ormal) Definition…

- How should we think about basic aggregates? What if we define them as a function?

$$S(X) = F(\{f(\tau(\{x_i\}))\})$$

Great! One important point…

$$f(X) = f(\{f(X_1), f(X_2) \ldots f(X_n)\})$$

Statistics		Basic Aggregates				
Type	**Formula**	$\sum x$	$\sum x^2$	$\sum xy$	$\sum y^2$	$\sum y$
Mean (avg)	$\frac{\sum x_i}{n}$	■				
Root Mean Square (rms)	$\sqrt{\frac{1}{n} \cdot \sum x^2}$		■			
Variance (var)	$\frac{\sum x_i^2 - n \cdot \text{avg}(x)^2}{n}$	■	■			
Standard Deviation (std)	$\sqrt{\frac{\sum x_i^2 - n \cdot \text{avg}(x)^2}{n}}$	■	■			
Sample Kurtosis (kur)	$\frac{1}{n}\sum(\frac{x_i - \text{avg}(x)}{\text{std}(x)})^4 - 3$	■	■			
Sample Covariance (cov)	$\frac{\sum x_i \cdot y_i}{n} - \frac{\sum x_i \cdot \sum y_i}{n^2}$	■		■		■
Simple Linear Regression (slr)	$\frac{\text{cov}(x,y)}{\text{var}(x)}, \text{avg}(x), \text{avg}(y)$	■	■	■		■
Sample Correlation (corr)	$\frac{n \cdot \sum x_i \cdot y_i - \sum x_i \cdot \sum y_i}{\sqrt{n \cdot \sum x_i^2 - (\sum x_i)^2}\sqrt{n \cdot \sum y_i^2 - (\sum y_i)^2}}$	■	■	■	■	■

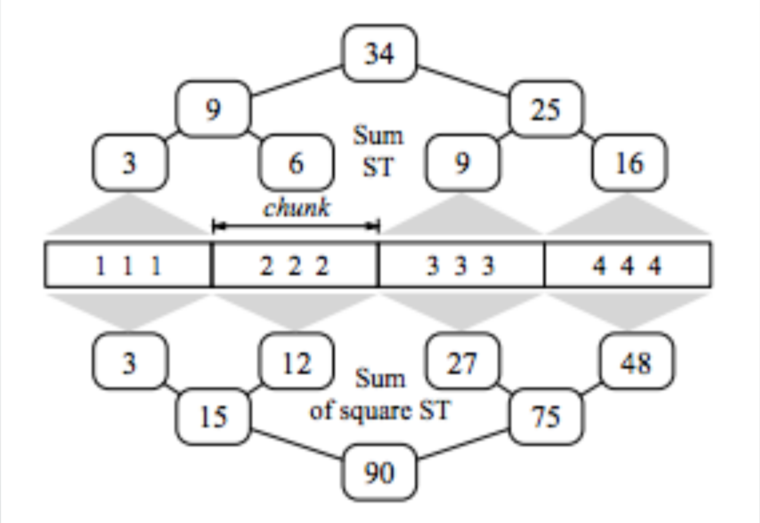
But is this Flexible Enough?

- Accounts for 90%+ of stats supported by Numpy and SciPy, 75%+ of stats supported by Wolfram.

How Often Does an Aggregate Form?

- 1 Chunk (of data)… For each chunk, one value exists FOR EACH basic aggregate type.
- If the granularity of a query (i.e. daily, weekly) doesn't match the granularity of a chunk, we scan at most the two surrounding chunks at the edges of the query range.

How are the Values Stored?



Segment Trees!

How Often Does a Segment Tree Form?

Per Column, Per Statistic

- Matches structure of queries
- Matches structure of aggregates

Major Benefit…

- Easily Parallelizable!
- Univariate: Divide columns between available threads.
- Multivariate: Independently build different segment trees for each combination of columns.

Operation Modes

- Offline
    - Data Canopy built in advance, library of basic aggregates available to start.
- Online
    - Data Canopy populated incrementally during query processing.
- Speculative
    - For a modest CPU/memory overhead to I/O tradeoff, incrementally construct more segment trees than those which are immediately needed.

Query Processing

- Map query range to set of chunks
  - If range fits chunks, synthesize result from basic aggregates
  - If residual range, compute basic aggregates for range
- Map a statistic to a set of basic aggregates

$$\{\{C\},[R_s,R_e),S\} \rightarrow \{\{C\},[c_s,c_e],R_d,\{f(\{\tau\})\},F\}$$

- Evaluate plan…
  - Offline Mode? No need to touch base data except to evaluate residual range.
  - Online/Speculative Mode? Form chunks associated with any residual range.

Query Cost

Term	Description
c	Number of columns
r	Number of rows
h	Number of chunks
s	Chunk size (bytes)
v_d	Record size (bytes)
v_{st}	ST node size (bytes)
#	Cache line size (bytes)

$$C_{syn} = C_{st} + C_r$$

$$C_{syn} = \frac{2 \cdot k \cdot s}{\#} + \left(2 \cdot b \cdot \log_2 \frac{r \cdot v_d}{s}\right)$$

How do these costs compare?

$$C_{scan} = \frac{R \cdot v_d}{\#}$$

R is the range of data. R_b is the point at which C_{scan} = C_{syn}

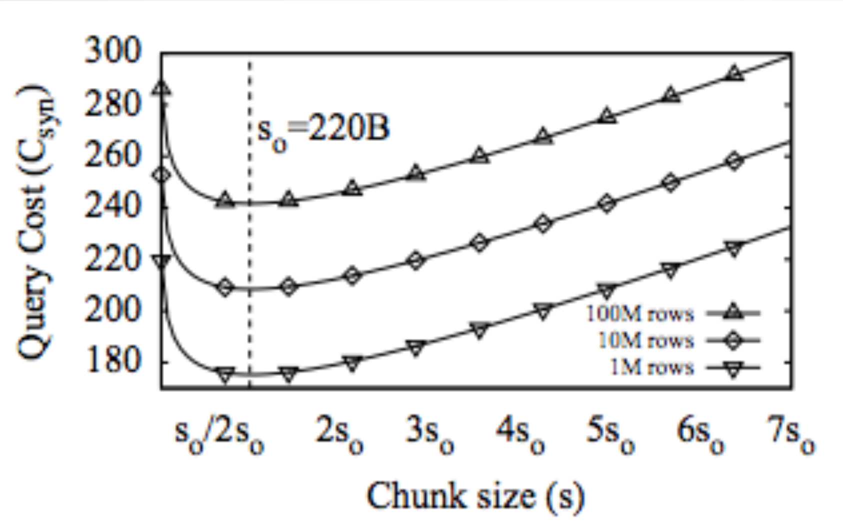
$$R_b = \frac{2 \cdot k \cdot s}{v_d} + \frac{2}{v_d} \cdot \# \cdot b \cdot \log_2 \left(\frac{r \cdot v_d}{s}\right)$$

So when R > R_b use the synthesized aggregates for optimal speed.

Selecting Chunk Size

- Dependent On… Hardware, type of requested statistic.

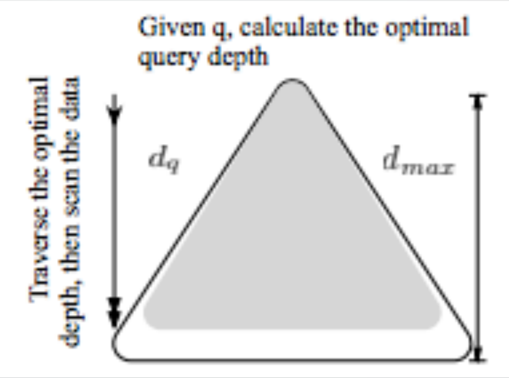
$$s_o = \frac{b \cdot \#}{k \cdot \ln 2}$$



Selecting Optimal Tree Search Depth

- This is based on the optimal size of a data chunk. When looking for answer to query, traverses segment tree to depth d_q. If answer not found, skips to scan data instead.

$$d_q = \log_2\left(\frac{r \cdot v_d}{s_q}\right)$$



What about Memory Size?

- Dependent on:
  - Types of statistical measures contained
  - Chunk size
  - Data size
- This raises a more interesting question…

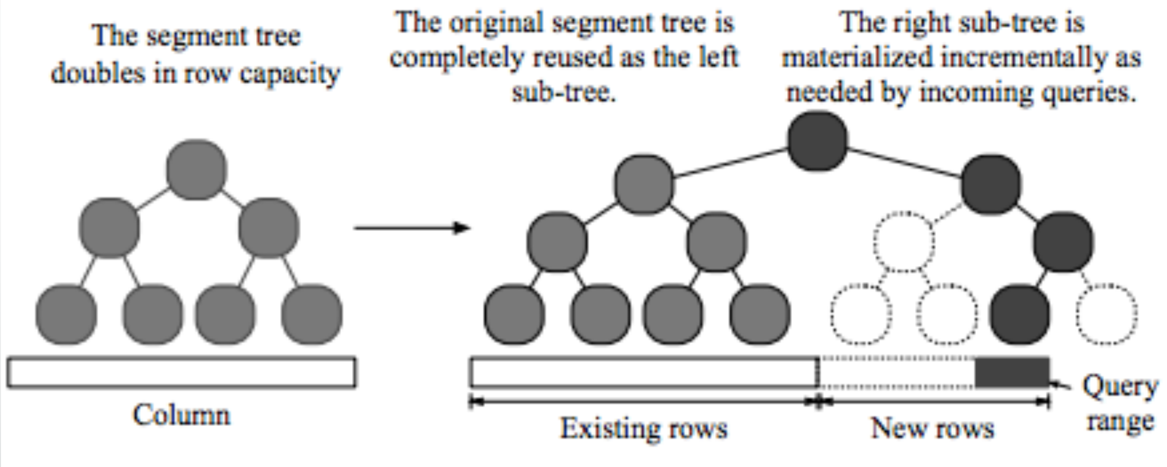
$$|DC(S)| = c \cdot v_{st} \cdot (2 \cdot \frac{r \cdot v_d}{s} - 1) \cdot \mathscr{F}(S)$$

How / When do we Evict?

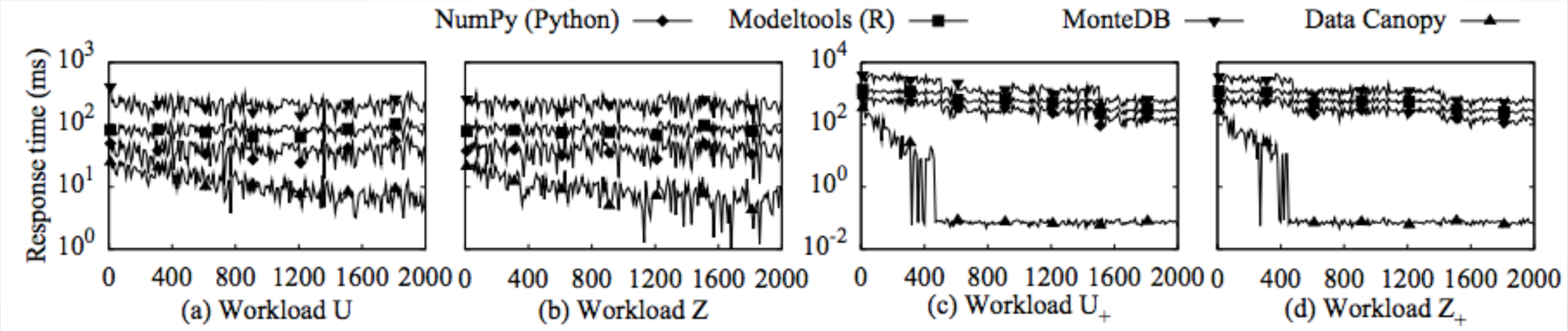
- Phase 1: Round-Robin removal of one layer of leaf nodes from every segment tree.
- Phase 2: Caches frequent data.
- Phase 3: Pushes whole segment tree to disk, keeps bitvector that marks any dirty chunks if the tree is later reloaded from disk.

Updating the Data Canopy

- To insert rows: When the capacity of the Data Canopy is reached, double the capacity of the segment trees by creating a new root.
- To insert columns: Simply add to types of trees Data Canopy can form.
- Updating Rows: Update old aggregate
- Deleting Rows: Decrement a counter on the chunk. Maintain invalidity segment tree.



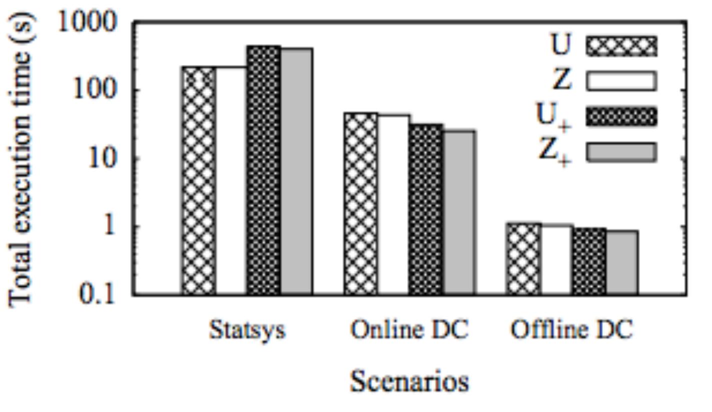
Experimental Analysis



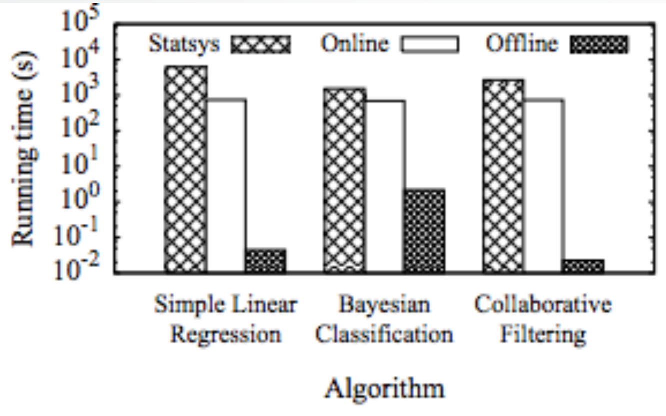
The longer the exploration path, the greater the benefit. Notice, as we increase in data repetition, we see improvements in performance. Perhaps the drop in c and d is due to generation of the Data Canopy, or switching of some of its policies.

Workload	Column Dist.	Range Size	Repetition
U	Uniform	*Unif(5,10) %*	low
Z	Zipfian	*Unif(5,10) %*	moderate
U_+	Uniform	Zoom-in	high
Z_+	Zipfian	Zoom-in	very high

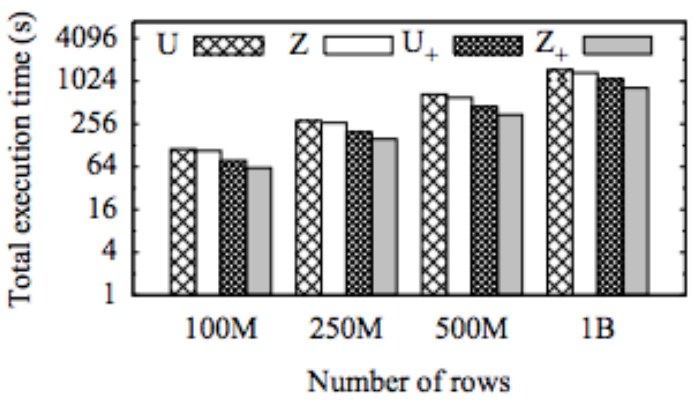
Experimental Analysis



Online and Offline Performance of DC

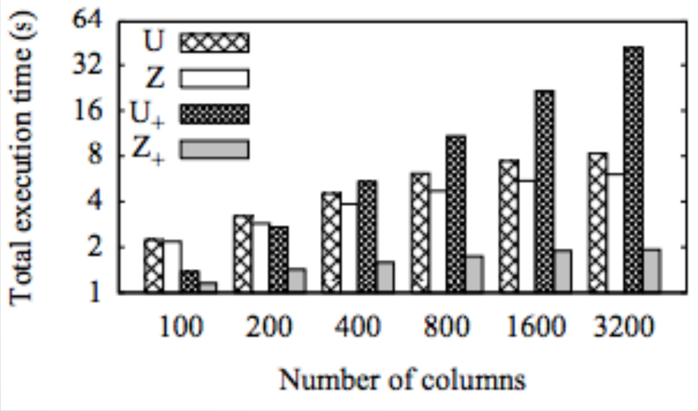


Speeding up ML Performance

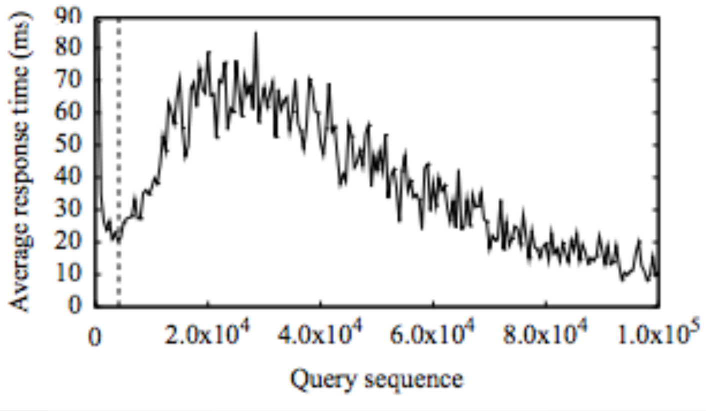


Linear Increase to Execution Time

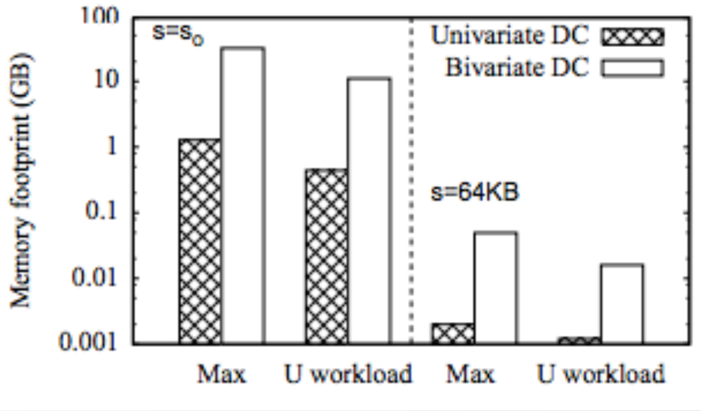
Experimental Analysis



Linear Increase to Execution Time



Rebound after Phase 2



Reduces memory footprint according to pressure. (Phase 1/Phase 2)

Thoughts

- A really well formulated paper, on a topic that is conceptually easy to grasp, but goes into a lot of depth.
- Could have expanded more on / tied together machine learning paradigms and examples of how they were constructed via Data Canopy aggregates.
- Would have liked to have known more concretely about when a phase is switched from one to the next in order to handle memory pressure.