Explorations in Interactive Visual Analytics
Supporting Analysis and Visualization At Scale

Doug Talbott
Bedarra Research Labs
doug.talbott@bedarra.com
What we’re going to talk about

• What is big data?
• Who is the big data user?
• What are we trying to accomplish in this space?
• Challenges we’ve encountered
• Things we’ve learned
What is big data?

Datasets that are too large or complex to be managed using traditional methods and tools…

**Volume**
33% of companies now working with 500TB or more

**Velocity**
63% of analytics will be real-time by 2015

**Variety**
80% of enterprise data is unstructured

Various sources including Gartner. IDC. Forrester
Who is the big data user?

• Domain experts in science, business, engineering, arts
  – A.K.A. ‘the data scientist’
  – Shortage of 150,000 -190,000 in US alone [McKinsey & Co]
  – Linear, lateral and critical thinking skills
  – Model-based programming skills

• Wide range of programming skills
  – 91% using R, SAS, Python, SQL*
  – 9% using Java, Hadoop (Pig, Hive, etc), SPSS, MatLab, Scala, C/C++*
  – Becoming familiar with clouds, clusters, grids, hadoop, etc

• Wide range of applications
  – Sentiment analysis, fraud detection, customer churn analysis, network monitoring, risk modelling, social network analysis, etc

*2014 KD Nuggets poll respondents
What are we trying to accomplish?

Trying to deliver a ‘small data’ user experience at ‘big data’ scale in terms of response times, interactivity and ease-of-use…
What are the challenges to delivering that kind of user experience at scale?

<table>
<thead>
<tr>
<th>Human Factors Challenges</th>
<th>Possible Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short attention spans</td>
<td>Increase processing and rendering speeds</td>
</tr>
<tr>
<td></td>
<td>Reduce latency</td>
</tr>
<tr>
<td>Loss of context</td>
<td>Increase processing and rendering speeds</td>
</tr>
<tr>
<td></td>
<td>Reduce latency</td>
</tr>
<tr>
<td>Limited short term memory</td>
<td>Keep UI controls, data, scripts visible</td>
</tr>
<tr>
<td></td>
<td>Increase processing and rendering speeds</td>
</tr>
<tr>
<td></td>
<td>Reduce latency</td>
</tr>
<tr>
<td>Increased cognitive load</td>
<td>Design UI controls to handle complexity</td>
</tr>
<tr>
<td></td>
<td>Flexible approaches to analysis</td>
</tr>
<tr>
<td>Increased perceptual load</td>
<td>Leverage pre-attentive processing skills</td>
</tr>
<tr>
<td></td>
<td>Encourage pattern matching</td>
</tr>
<tr>
<td></td>
<td>Flexible approaches to visualization</td>
</tr>
</tbody>
</table>
A typical analysis workflow example

• **VAST Challenge Case Study**
  – Analyse cell phone records to identify a social network of criminals
  – Cell phone records contain source, dest, duration, tower, timestamp
  – Phones are numbered 0 to 400
  – Original set was tiny so we ‘poofed’ it up to 35 million records
  – Suspicious behaviors to look for…
    • Destinations receiving lots of calls from different sources
    • Destinations suddenly disappear; destinations suddenly appear and receive lots of calls from the same set of different sources

• **Technology**
  – Ivy workbench, but other solutions may do similar things (e.g. R Studio, Rapid Miner; Datawatch, Weka, Tableau, Pentahoe, Palantir; SpotFire)
  – Internal server using 4 cores
```sql
function load35millionrecords()
    cellRecords : 35000000 # (["IID": enlist ","] 0: `:/CARE4/ExampleData/cells.csv;
    // produce basic summary table
    gotoPresentation.summarize(cellRecords);
    // find the top five people who called the most
    0:select[5] from 'counts xdesc select counts: count source by dest from cellRecords;
    // find the top ten people who called other people
    0:select[5] from 'counts xdesc select counts: count dest by source from cellRecords;
    // find the top ten people who talked the most
    0:select[5] from 'talk xdesc select talk: sum duration by source from cellRecords;
    // Create a 400x400 matrix of called and calling parties
    matrix : value [400x400; ++i] each dict;
    colStochastic: flip matrix % sum each matrix;
    v: (count colStochastic) # 1;
    rankings : [x % sum x] [x mmu y](colStochastic)/[v];
    rankedIDs : (key dict) rankings;
    keyCellPhones: key desc rankedIDs;
    // create graph of the most important people as destinations
    0:select distinct source by dest from cellRecords where dest in 13#keyCellPhones;
```
So what did we see in the example?

- **Real-time ‘tumbling of data’ to explore and find relationships in the data**
  - Very fast loading of records
  - Real-time analysis and visualization of 35 million records
  - Summary, page rank, and query scripting ‘on-the-fly’
  - Different plot types
  - Different column configurations
  - Different levels of granularity (bin sizing)
  - Different levels of detail (zooming)

- **Minimal human factors impediments**
  - No attention span problems (fast processing)
  - No loss of context (fast processing)
  - No memory problems (visible controls, scripts, data)
  - Reduced cognitive load (flexible, scalable control interfaces)
  - Reduced perceptual load (binning to leverage pre-attentive processing)
What are the key learnings so far?

• **Supporting analysis at scale**
  – Support integration of analysis and visualization tasks
  – Support an interactive, non-linear task flow at scale

• **Supporting visualization at scale**
  – Leverage server-side computing and rendering
  – Leverage techniques that encourage pattern matching
  – Allow users to deviate from accepted visual design rules

• **Support interaction at scale**
  – Manage the data-to-ui ratio
  – Provide interaction control to support data complexity
  – Make scripting a primary task
Support integration of analysis and visualization tasks
Need to tightly couple analysis and visualization

- Schneiderman’s Data Visualization Mantra
  - “Overview first, zoom & filter, details on demand”
- Visualization tasks (blue) merged with analysis tasks (orange)
- Realistically you want to do these tasks in almost any order

<table>
<thead>
<tr>
<th>Analysis and Visualization Tasks</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extract/load</td>
<td>Only take records from Denmark out of the dataset</td>
</tr>
<tr>
<td>Transform or lookup/load</td>
<td>Change gender from male, M, MALE, Male to zero</td>
</tr>
<tr>
<td>Script</td>
<td>Write a query, write a cluster analysis routine</td>
</tr>
<tr>
<td>Visualize (high-level, details)</td>
<td>View data as scatter chart</td>
</tr>
<tr>
<td>Filter</td>
<td>All records where sex equals female</td>
</tr>
<tr>
<td>Sort, categorize, relate, organize, mine</td>
<td>Rank children by height in grade 2</td>
</tr>
<tr>
<td>Brush, highlight, telescope</td>
<td>Rollover to see record details</td>
</tr>
<tr>
<td>Drill, zoom, pan</td>
<td>Rubberband select a region and zoom in</td>
</tr>
<tr>
<td>Annotate</td>
<td>Add a comment to a visualization</td>
</tr>
<tr>
<td>Collaborate</td>
<td>Sharing the visualizations, filters and computations</td>
</tr>
</tbody>
</table>
Support an iterative, non-linear taskflow at scale
Users need to be able to ‘tumble’ their data in order to explore and find relationships.

### Organization Techniques

<table>
<thead>
<tr>
<th>Location</th>
<th>Alpha</th>
<th>Time</th>
<th>Category</th>
<th>Heirarchy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maps</td>
<td>Tables, Pivot, Text</td>
<td>Timelines</td>
<td>Pie, Bar, Line, Scatter, Heatmap, Histogram</td>
<td>Decision Tree, Treemap, Graphs</td>
</tr>
</tbody>
</table>

### Analysis Techniques

**Summarization**
- Keyword analysis

**Clustering**
- K-means

**Association Rules**
- Shopping cart analysis

**Anomaly Detection**
- Outliers, link analysis

**Classification**
- Groupings

**Regression**
- Statistical models

### Rankings
- Parts-to-whole
- Across time
- Deviations
- Distributions
- Correlations
Leverage server-side processing and rendering to improve interactivity at scale
Server-side processing lets you query and compute quickly...

100TB
NYSE TAQ
5000 days
1.1 trillion quotes
65 billion trades

Hardware
16 core 256GB
Date partitioned
Sym indexed
Queries in RAM

Small day query 18ms
2003.09.10. 35M quotes. 5M trades.
select last bid by sym from quote where date=d, sym in S

Big day query 290 ms
2013.04.03. 610M quotes. 30M trades.
select last bid by sym from quote where date=d, sym in S
Server-side rendering lets you visualize faster

- Tried rendering on client first
- Client is for interacting
- Server is for processing and rendering
- Images are shipped from server to client
- Data brushing is handled by shipping clicks to server
A few numbers…

- **Rendering 1 million points**
  - Client: JS, D3, SVG, Canvas objects ~22 seconds (Apple)
  - Server: PNG shipped to client ~ 4 seconds (Vanilla Amazon cloud)

- **Rendering 10 million points**
  - Client: JS, D3, SVG, Canvas -- Browser Screen of Death (Apple)
  - Server: PNG shipped to client ~28 seconds (Vanilla Amazon cloud)

- **Rendering points using binning**
  - Rendering 1 million ~ 1 second (Vanilla Amazon cloud)
  - Rendering 10 million ~ 4 seconds (Vanilla Amazon cloud)
  - Rendering 100 million ~22 seconds (Internal 4 cores)
  - Rendering 500 million ~116 seconds (Internal 4 cores)
Leverage techniques that encourage pattern finding and matching
Pattern matching techniques already exist and can be applied to big data

- **Huge range of techniques already exist**
  - Aggregation (e.g. count, sum, average, min, max, etc)*
  - Summarization (e.g. keyword extraction)*
  - Clustering (e.g. k-means, graph/link analysis)*
  - Anomaly detection (e.g. outlier, change, deviation, graph/link analysis)*
  - Association rules (e.g. shopping basket)

- **Many of these techniques can:**
  - leverage our ability to pre-attentively process data
  - be performed at scale ‘on-the-fly’
  - be applied to different visualization types (e.g. bar, line, box plots)
  - replace or be augmented by visualization (e.g. page rank analysis)
Pattern matching leverages our ability to pre-attentively process information

- Huge body of work on pre-attentive processing (Bertin, Mackinlay)
- Without paying attention, people notice certain visual attributes

**Quantitative Data**
1. Position
2. Length
3. Angle (slope)
4. 2D Area/ 3D Volume (size)
5. Density (opacity)
6. Color Saturation
7. Color Hue
8. Texture (not applicable)
9. Connection (not applicable)
10. Containment (not applicable)
11. Shape (not applicable)

**Length**
- Last item is twice as big as the others

**Shape**
- Last one is different but cannot tell how it is different

---

Automating the design of graphical representations of relational information J.D. Mackinlay 1986
High-speed visual estimation using preattentive processing, C.G. Healey, K.S. Booth, J.T. Enns 1996
Seminology of graphics, J. Bertin, 1967
Aggregation allows you to identify visual patterns

- Change bin granularity
- Change type of aggregation
- Apply to area, density, color
- Identify visual patterns
Applied to geomap

Where you don’t want to live in Sacramento

- 10M crimes (20 x 20 bins rendered ~3 seconds)
- Bin count applied to opacity attribute
Applied to scatter

- 10M crimes (20x20 bins) renders in ~2 seconds
- Bin count applied to size attribute
Applied to heatmap

- 10M crimes (20 x 20 bins rendered in ~2 seconds)
- Bin count applied to opacity attribute
Applied to histogram

- 10M crimes by police zone (19 bins rendered in ~2 seconds)
- Bin count applied to length attribute
Plotting ‘all the data’ can actually hide the data

- 1,005,000 points between zero and one using opacity attribute
- Plotting all the points took ~5 seconds and produced no pattern
Plotting by bins actually reveals the pattern and it’s faster

- Same graph plotted with 200x200 bins in ~2 seconds
- Now you can see the pattern
Patterns might even let you find a needle in a haystack

- Distributed patterns appear to hold at 2000:1,000,000

- Concentrated patterns appear to hold at 250:1,000,000
Allow users to ‘bend’ the rules of good graph design
‘Analysing’ rules may need to be different than ‘viewing’ rules

- Tufte developed a great set of design rules for viewing graphs
- This terrible graph violates several of his rules

“Tell the truth with your data” and “Show your data”

“Erase non-data ink”
However… at scale sometimes
“showing the data” can hide the data

• One million points in ~5 seconds
• No pattern
Not ‘showing all the data’ can reveal a pattern

- One million points in ~5 seconds
- Rendering at 5% opacity yields pattern
Not showing all the data, and it’s ugly, but it works!

- Manipulating the RGB levels removes points and enhances pattern
- Can be done completely in client using JS image libraries
Sometimes ‘telling the truth with your data’ provides little value

- The Rule: Plot Y axis from zero so data is not exaggerated
- 1000 Y bars between 50,000 and 50,050*
- When plotted from zero, this plot is kind of useless

* One Y value is plotted at 0,0 to trick our renderer
Plotting from min and max values at least shows range

- Same points plotted from min (50,000) and max (50,050) range
- When plotted by range, we can see a bit of a gap
- Let the analyst decide what is important
Sometimes ‘too much data’ can actually show you things

• The rule: Don’t use more than 7 +/- 2 legend items
• However, this example contains 80 legend items
  – Occurences of top 80 keywords in Moby Dick (Y) by chapter (X)
  – Easily see chapters are largely oriented on single keywords
  – Easily see a few chapters contain no keywords
Too much data can always be managed

- Filtering out noise is handled interactively (e.g. rollovers)
- Example of highlighting Jonah’s importance across chapters
Maximize the plot-to-ui ratio
Striking the balance between ui control area and plot area

- Big data plots often require lots of pixels for pattern identification
- But small data UIs allocate quite a bit of the screen to ui controls
- Need to strike better balance between ‘ui controls’ and ‘plot’
Controls are shown but scroll so we can maximize plot size

- Analysts use controls constantly
- Tradeoff between control accessibility and plot size
Controls need to support data complexity
Rethinking standard UI controls at scale

- Standard controls work for few columns, small ranges and iterations
- Tedious with more columns, broader ranges, and lots of iterations
- Tradeoff between ease of learn and expressiveness

Easy to learn
Tedious to use with lots of columns, ranges, iterations

Enter query:
```
select from dataset where source in (1),
dest in (0 5 306 309)
```

Easier to use with lots of columns, ranges, iterations
Harder to learn
Rethinking standard UI interfaces at scale

• Same issue with general query interfaces
  – Node & link, tree, spreadsheet, text interface styles

• Opted for spreadsheet style as an 80% solution
  – Harder to learn but easier to express complex queries
  – Handles the majority of queries

Node and link style  Spread sheet style  Text style

select from crime where ((beat = '3C) and (grid =1115)) or ((beat = '2A)) or ((district =1)) or ((district =2)) or ((district >3) and (district <4))

Easy to learn
Less expressive and flexible
Real estate issues

Harder to learn
More expressive and flexible
Syntax issues
Small data interfaces tend to auto update…
Big data interfaces need to manually update

“Please wait while I replot 200 million points to change the color from blue to red…”

“Sigh… But I wanted to remove the background grid and change the bin size too!”
Small data interfaces tend to auto execute…
Big data interfaces need to manually execute

- Small data UIs execute operations automatically
- Big data UIs should execute operations manually
- Big data UIs should provide ‘examples’ and ‘estimates of results’ first
- Same principles can be applied to any large operation

Estimate of matches
Example of the results
Make scripting a primary task
Make scripting a primary task

• **Scripting can be applied to all operations**
  – Extract, translate, load (ETL) operations
  – Data mining techniques (models and analysis)
  – Querying

• **Pros**
  – Efficient, expressive and flexible

• **Cons**
  – Learning curve required (increased memory and cognitive load)
  – Many analysts are familiar with a language (e.g. R, SQL, MatLab)
A little scripting can go a long way…

- Via interface: 500 million 20x20 bins ~116 seconds
- Via scripting: 500 million 20x20 bins ~27 seconds

```sql
1 select avg profit, avg sale by 5 xbar orderQty, region from sales
```
A little more scripting can make the difference between the right and wrong answer

1. Visual analysis allowed us to identify key players

2. Markov cluster analysis revealed nothing

3. Page rank analysis revealed additional key players

4. Page rank analysis augmented by directed graph visualization
But if you are going to script...

- Allow scripting ‘on-the-fly’
- Provide a continuous feedback loop
- Allow users to create their own languages (e.g. DSLs)
- Allow users to move freely between scripting and visualizing
What we covered...

• **Objective**
  – Provide a ‘small data’ experience at ‘big data’ scale in terms of response times, interactivity and ease-of-use

• **Challenges**
  – Address human issues (e.g. short term memory, loss of context, etc)

• **Learnings**
  – Support integration of analysis and visualization tasks
  – Support an interactive, non-linear task flow at scale
  – Leverage server-side computing and rendering
  – Leverage techniques that encourage pattern matching
  – Allow users to deviate from good visual design rules
  – Manage the data-to-ui ratio
  – Provide interaction control to support data complexity
  – Make scripting a primary task
Thank you!

Doug Talbott
Bedarra Research Labs
doug.talbott@bedarra.com
www.bedarra.com