Data Visualization (DSC 530/CIS 568)

Data Wrangling

Dr. David Koop
Attribute Aggregation

- Remember reducing attributes—use statistics: either one variable matches another or doesn't change!
- We can also use similar criteria for aggregating attributes
- **Cluster** similar attributes together
  - How?
Principle Component Analysis (PCA)

original data space

Gene 1
Gene 2
Gene 3

PC 1
PC 2

PCA

component space

Gene 2
Gene 1

PC 1
PC 2

[M. Scholz, CC-BY-SA 2.0]
Here's the plot of the data along the first principal component. Already we can see something is different about Northern Ireland.

Now, see the first and second principal components, we see Northern Ireland a major outlier. Once we go back and look at the data in the table, this makes sense: the Northern Irish eat way more grams of fresh potatoes and way fewer of fresh fruits, cheese, fish and alcoholic drinks. It's a good sign that structure we've visualized reflects a big fact of real-world geography: Northern Ireland is the only of the four countries not on the island of Great Britain. (If you're confused about the differences among England, the UK and Great Britain, see: this video.)

For more explanations, visit the Explained Visually project homepage. Or subscribe to our mailing list.
Non-linear Dimensionality Reduction

Original data space $\mathcal{X}$

Component space $\mathcal{Z}$

$\Phi_{gen} : \mathcal{Z} \rightarrow \mathcal{X}$

$\Phi_{extr} : \mathcal{X} \rightarrow \mathcal{Z}$

[M. Scholz, CC-BY-SA 2.0]
**Probing Projections: Attribute Heatmap**
Probing Projections: Projection Errors

Fig. 6. Halos represent the cumulative error for the respective samples. White indicates that a majority of samples is more similar than indicated by their distance to the given sample; grey indicates the opposite. The paths travelled by the points are shown as lines, leading from the points' original positions in the projection to the new, corrected positions (see Figure 8). This connects them to their original positions in the projection, and displays the size of the distance error at the same time. Resembling the brightness encoding of the halos, the brightness of the lines indicates whether they've moved closer or farther away.

A problem with this solution is that it introduces new distortions in the spatial relationship between all other points. Only the distances directly between the selected point and the other points are reliable, whereas all the other distances are distorted, and the new positioning might lead to wrong assumptions about potential clusterings. To mitigate this problem, the correction paths are shown.

Another solution would be to recompute the projection while preserving the distances from and to the selected point and being more generous with distance errors among the remaining points. This would somewhat reduce the introduced distortions. However, in a recomputed projection, the positions of the points might change significantly, most likely leading to completely different positions for all points, possibly confusing the observer even if an animation is used.

Fig. 7. Dendrograms mapped onto the projection. Left: projection with low projection error. Right: high projection error.

4.5.3 Dendrogram

In addition to the visualization of errors and corrections, a dendrogram can visualize the samples with regard to their position in the clustering hierarchy. Such a dendrogram (using the same agglomerative algorithm as the clusters) overlaid onto the projection may also help to visualise high-dimensional distances on the projection space. It graphically emphasises clusters by connecting close dots through dense lines. Interestingly, the dendrogram is a surprisingly good indicator of goodness of fit: if many thick, long lines intersect, it is likely that the projection is of low quality.

EXAMPLE: OECD COUNTRIES

To illustrate the functionality of the interface we visualize the dataset of OECD countries in the prototype (see Figure 9). The dataset contains 8 dimensions for 36 countries. First, the viewer is drawn to the projection and notices Turkey that seems to be a clear outlier, far away from all other countries. To explore why this is, the viewer can examine this sample by hovering over it. A tooltip relating Turkey to the rest of the dataset appears, showing that it deviates strongly from the mean in nearly every dimension. This indicates the positioning as outlier is probably correct.

To test this assumption and build up trust in the visualization, the viewer selects 'correct distances', showing the high-dimensional distances between Turkey and the other countries. This reveals that Turkey should be even farther apart from several of the other countries. Having confirmed that Turkey is an outlier in this dataset, the viewer uses the built-in clustering to get a sense of how the countries are grouped. Playing around with the number of clusters, they notice that there seem to be seven clusters roughly corresponding to the geographical and geopolitical placement of the countries.

Taking a closer look at the positioning of the clustered countries, they realise that the arrangement seems to roughly correspond to geographic directions: Northern and Southern countries are roughly distributed along the vertical axes, East and West along the horizontal. To find out if or how this correlates with the dimensions, the viewer first compares the different clusters. Here the differences along the dimensions are very much pronounced. Interestingly though, life expectancy is lower in Latin America than Asia, while the self-reported health is higher for the former than the latter.

After a few more comparisons between the clusters, the viewer becomes interested in the dimension life satisfaction and turns towards the heatmaps. They notice that the values for life satisfaction and self-reported health seem to be higher in the Western countries, whereas the value for employees working very long hours seems to be especially high in the countries of the far East and the South.

[http://www.oecdbetterlifeindex.org/]

[J. Stahnke et al., 2015]
Focus+Content Overview

→ **Embed**
  
  ➔ **Elide Data**
  
  ➔ **Superimpose Layer**
  
  ➔ **Distort Geometry**

→ **Reduce**

  ➔ **Filter**
  
  ➔ **Aggregate**
  
  ➔ **Embed**

[Munzner (ill. Maguire), 2014]
Elision: DOITrees

- Example: 600,000 node tree
  - Multiple foci (from search results or via user selection)
  - Distance computed topologically (levels, not geometric)

[Heer and Card, 2004]
Superimposition with Interactive Lenses

(a) Alteration
(b) Suppression

[ChronoLenses and Sampling Lens in Tominski et al., 2014]
Distortion

It can be difficult to observe micro and macro features simultaneously with complex graphs. If you zoom in for detail, the graph is too big to view in its entirety. If you zoom out to see the overall structure, small details are lost.

Focus + context techniques allow interactive exploration of an area.
Distortion Concerns

• Distance and length judgments are **harder**
  - Example: Mac OS X Dock with Magnification
  - Spatial position of items changes as the focus changes

• Node-link diagrams not an issue… why?

• Users have to be made aware of distortion
  - Back to scatterplot with distortion example
  - Lenses or shading give clues to users

• **Object constancy**: understanding when two views/frames show the same object
  - What happens under distortion?
  - 3D Perspective is distortion… but we are well-trained for that

• Think about **what** is being shown (filtering) and method (fisheye)
Assignment 4

- Filtering: Add population filter
- Aggregation: Add histogram of percentage of subsidized housing
- Brushing: Filter the map based on histogram
Projects

• Deadlines:
  - Presentations on May 6, 3-6pm
  - Final Reports not late until the end of the day on the 6th

• Design feedback was posted Friday
  - Good starts, some designs very complete
  - Usually no scrolling from one view to the next for a multiple-view visualization, try to fit within a screen
  - Provide complete overviews, even if interactions will filter views or provide details
Data Wrangling

• Problem 1: Visualizations need data
  • Solution: The Web!

• Problem 2: Data has extra information I don't need
  • Solution: Filter it

• Problem 3: Data is dirty
  • Solution: Clean it up

• Problem 4: Data isn't in the same place
  • Solution: Combine data from different sources

• Problem 5: Data isn't structured correctly
  • Solution: Reorder, map, and nest it
Hosting data

- github.com
- gist.github.com
- figshare.com
- myjson.com
- Other services
Why JavaScript?

- Python and R have great support for this sort of processing
- Data comes from the Web, want to put visualizations on the Web
- Sometimes unnecessary to download, process, and upload!
- More tools are helping JavaScript become a better language
JavaScript Data Wrangling Resources

- https://.observablehq.com/@dakoop/learn-js-data
- Based on http://learnjsdata.com/
- Good coverage of data wrangling using JavaScript
Comma Separated Values (CSV)

• File structure:

```plaintext
cities.csv:

<table>
<thead>
<tr>
<th>city</th>
<th>state</th>
<th>population</th>
<th>land area</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>WA</td>
<td>652405</td>
<td>83.9</td>
</tr>
<tr>
<td>new york</td>
<td>NY</td>
<td>8405837</td>
<td>302.6</td>
</tr>
<tr>
<td>boston</td>
<td>MA</td>
<td>645966</td>
<td>48.3</td>
</tr>
<tr>
<td>kansas city</td>
<td>MO</td>
<td>467007</td>
<td>315.0</td>
</tr>
</tbody>
</table>
```

• Loading using D3:

```javascript
let d3 = require('d3');
d3.csv('./data/cities.csv').then((data) => {
  console.log(data[0]);
});
```

• Result:

```javascript
=> {city: "seattle", state: "WA", population: 652405, land area: 83.9}
```

• Values are strings! Convert to numbers via the unary + operator:

```javascript
-d.population => "652405"
+d.population => 652405
```

[http://learnjsdata.com]
Tab Separated Values (TSV)

• File structure:

```
animals.tsv:

name  type    avg_weight
---  ------    -------
tiger mammal  260
hippo mammal  3400
komodo dragon reptile 150
```

• Loading using D3:

```
d3.tsv("/data/animals.tsv").then(function(data) {
  console.log(data[0]);
});
```

• Result:

```javascript
=> {name: "tiger", type: "mammal", avg_weight: "260"}
```

• Can also have other delimiters (e.g. '|', ';')

[http://learnjsdata.com]
JavaScript Object Notation (JSON)

• File Structure:

```javascript
employees.json:
[
  {
    "name": "Andy Hunt",
    "title": "Big Boss",
    "age": 68,
    "bonus": true
  },
  {
    "name": "Charles Mack",
    "title": "Jr Dev",
    "age": 24,
    "bonus": false
  }
]
```

• Loading using D3:

```javascript
d3.json("/data/employees.json").then(function(data) {
  console.log(data[0]);
});
```

• Result:

```javascript
=> {name: "Andy Hunt", title: "Big Boss", age: 68, bonus: true}
```
Loading Multiple Files

- Use Promise.all to load multiple files and then process them all

```javascript
Promise.all([d3.csv("/data/cities.csv"),
    d3.tsv("/data/animals.tsv")])
    .then(analyze);

function analyze(data) {
    cities = data[0]; animals = data[1];

    console.log(cities[0]);
    console.log(animals[0]);
}
=> {city: "seattle", state: "WA", population: "652405", land area: "83.9"}
{name: "tiger", type: "mammal", avg_weight: "260"}
```
Combining Data

- Suppose given products and brands
- Brands have an id and products have a brand_id that matches a brand
- Want to join these two datasets together
  - Product.brand_id => Brand.id
- Use a nested forEach/filter
- Use a native join command

[http://learnjsdata.com]
Summarizing Data

• d3 has min, max, and extent functions of the form
  - 1st argument: dataset
  - 2nd argument: accessor function

• Example:
  ```javascript
  var landExtent = d3.extent(data, function(d) {
    return d.land_area;
  });
  console.log(landExtent);
  => [48.3, 315]
  ```

• Summary statistics:
  - mean, median, deviation
  - Same format

• Median Example:
  ```javascript
  var landMed = d3.median(data, function(d) {
    return d.land_area;
  });
  console.log(landMed);
  => 193.25
  ```
Nesting Data

- Take a flat structure and turn it into something nested
- Often similar to a groupby in databases
- `key` indicates groupings
- `rollup` indicates how the groups are processed/aggregated
- Last function specifies the data and how the output should look
  - `entries`: `[{key: <key>, value: <value>}]`
  - `object`: `{<key>: <value>, ...}`
  - `map`: `{<key>: <value>, ...}` but as a `d3.map` (safer than object, but uses `get/set` instead of square brackets `[]`)
Nesting Example

• Data

```javascript
var expenses = [{"name":"jim","amount":34,"date":"11/12/2015"},
{"name":"carl","amount":120.11,"date":"11/12/2015"},
{"name":"jim","amount":45,"date":"12/01/2015"},
{"name":"stacy","amount":12.00,"date":"01/04/2016"},
{"name":"stacy","amount":34.10,"date":"01/04/2016"},
{"name":"stacy","amount":44.80,"date":"01/05/2016"}];
```

• Using d3.nest:

```javascript
var expensesAvgAmount = d3.nest()
    .key(function(d) { return d.name; })
    .rollup(function(v) { return d3.mean(v, function(d) { return d.amount; }); })
    .entries(expenses);

console.log(JSON.stringify(expensesAvgAmount));
```

• Result:

```javascript
=> [{"key":"jim","values":39.5},
    {"key":"carl","values":120.11},
    {"key":"stacy","values":30.3}]
```

[http://learnjsdata.com]