Data Visualization (CIS 468)

Data Wrangling

Dr. David Koop
Dimensionality Reduction

• Attribute Aggregation: Use fewer attributes (dimensions) to represent items

• Combine attributes in a way that is more instructive than examining each individual attribute

• Example: Understanding the language in a collection of books
  - Count the occurrence of each non-common word in each book
  - Huge set of features (attributes), want to represent each with an aggregate feature (e.g. high use of "cowboy", lower use of "city") that allows clustering (e.g. "western")
  - Don't want to have to manually determine such rules

• Techniques: Principle Component Analysis, Multidimensional Scaling family of techniques
PCA: 17 dimensions to 2

<table>
<thead>
<tr>
<th>Category</th>
<th>England</th>
<th>N Ireland</th>
<th>Scotland</th>
<th>Wales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alcoholic drinks</td>
<td>375</td>
<td>135</td>
<td>458</td>
<td>475</td>
</tr>
<tr>
<td>Beverages</td>
<td>57</td>
<td>47</td>
<td>53</td>
<td>73</td>
</tr>
<tr>
<td>Carcase meat</td>
<td>245</td>
<td>267</td>
<td>242</td>
<td>227</td>
</tr>
<tr>
<td>Cereals</td>
<td>1472</td>
<td>1494</td>
<td>1462</td>
<td>1582</td>
</tr>
<tr>
<td>Cheese</td>
<td>105</td>
<td>66</td>
<td>103</td>
<td>103</td>
</tr>
<tr>
<td>Confectionery</td>
<td>54</td>
<td>41</td>
<td>62</td>
<td>64</td>
</tr>
<tr>
<td>Fats and oils</td>
<td>193</td>
<td>209</td>
<td>184</td>
<td>235</td>
</tr>
<tr>
<td>Fish</td>
<td>147</td>
<td>93</td>
<td>122</td>
<td>160</td>
</tr>
<tr>
<td>Fresh fruit</td>
<td>1102</td>
<td>674</td>
<td>957</td>
<td>137</td>
</tr>
<tr>
<td>Fresh potatoes</td>
<td>720</td>
<td>1033</td>
<td>566</td>
<td>874</td>
</tr>
<tr>
<td>Fresh Veg</td>
<td>253</td>
<td>143</td>
<td>171</td>
<td>265</td>
</tr>
<tr>
<td>Other meat</td>
<td>685</td>
<td>586</td>
<td>750</td>
<td>803</td>
</tr>
<tr>
<td>Other Veg</td>
<td>488</td>
<td>355</td>
<td>418</td>
<td>570</td>
</tr>
<tr>
<td>Processed potatoes</td>
<td>198</td>
<td>187</td>
<td>220</td>
<td>203</td>
</tr>
<tr>
<td>Processed Veg</td>
<td>360</td>
<td>334</td>
<td>337</td>
<td>365</td>
</tr>
<tr>
<td>Soft drinks</td>
<td>1374</td>
<td>1504</td>
<td>1572</td>
<td>1256</td>
</tr>
<tr>
<td>Sugars</td>
<td>156</td>
<td>139</td>
<td>147</td>
<td>175</td>
</tr>
</tbody>
</table>

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.

Tasks in Understanding High-Dim. Data

Task 1
In HD data → Out 2D data

What?
- In High-dimensional data
- Out 2D data

Why?
- Produce
- Derive

Task 2
In 2D data → Out Scatterplot Clusters & points

What?
- In 2D data
- Out Scatterplot
- Out Clusters & points

Why?
- Discover
- Explore
- Identify

How?
- Encode
- Navigate
- Select

Task 3
In Scatterplot Clusters & points → Out Labels for clusters

What?
- In Scatterplot
- In Clusters & points
- Out Labels for clusters

Why?
- Produce
- Annotate

[Munzner (ill. Maguire), 2014]
Focus+Content Overview

- **Embed**
  - Elide Data
  - Superimpose Layer
  - Distort Geometry

- **Reduce**
  - Filter
  - Aggregate
  - Embed

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

- 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
Assignment 4

• Link
• Interaction, Network, and Multiple Views
• Due Wednesday
• Questions?
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://beta.observablehq.com/@dakoop/learn-js-data](https://beta.observablehq.com/@dakoop/learn-js-data)
• Based on [http://learnjsdata.com/](http://learnjsdata.com/)
• Good coverage of data wrangling using JavaScript
Comma Separated Values (CSV)

• File structure:

```javascript
let cities = ['seattle', 'WA', 652405, 83.9],
  newYork = ['new york', 'NY', 8405837, 302.6],
  boston = ['boston', 'MA', 645966, 48.3],
  KansasCity = ['kansas city', 'MO', 467007, 315.0];
```

• Loading using D3:

```javascript
d3.csv('/data/cities.csv').then(function(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
```
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:

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

• Can also have other delimiters (e.g. '|', ';')
JavaScript Object Notation (JSON)

- File Structure:

  employees.json:
  
  ```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
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** indicate 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}]
```
d3-array 2.0

- Works with iterables
- group and rollup are separate now