Evaluating Input Data

Evaluating Input Data

- **Data formats**
- Data quantity
- Data quality

Data Formats

- **Data comes in many formats**
  - Format: the structure and encoding used to represent information
- **Format is primarily important at three points in the process**
  1. Format of the data provided to you or collected by you
  2. Format used as input to the analysis
  3. Format produced as output from the analysis
- **Some formats are better suited to certain uses than the others**
- **Data is sometimes converted to other formats during processing**
- **Some formats map to a relational model more than others**

Log Files

- **Log files are generated by applications and devices**
  - Examples: Web servers, mail servers, Hadoop, cell phones
- **Data scientists view logs as a valuable source of information**
  - Contain data that's too expensive to store in a transactional DB
  - Data is available immediately – no need to wait for ETL process
  - Log analysis does not require putting load on production system

```
$ head -n1 httpd.log
192.168.5.137 - - [17/Aug/2012:21:18:36 -0600]
"GET /products/widget.jsp?sku=16879 HTTP/1.1" 200 8472
"http://www.example.com" "Mozilla/5.0 (Windows NT 5.1; rv:2.0) Gecko/
20100101 Firefox/4.0" "uid=jsmith;usertype=Customer;region=midwest"

$ cut -f1 -d' ' httpd.log | sort | uniq -c | sort -rn | head -n2
 283  192.168.5.137
  79  10.9.8.47
```

Fixed-Width and Delimited Text Files

- **Data is sometimes provided as fields in text files**
  - Common for data exported from databases or spreadsheets
  - Typically one record per line
- **Two main variants**
  - Fixed-width: field starts at position M and occupies N characters
  - Delimited: fields separated by characters such as comma or tab
- **CSV files can be deceptively difficult to parse**
  - There is a specification, but few follow it exactly
  - Variations on quoting, embedded commas, missing fields, etc.

```
$ cut -c10-14 fixedwidth.txt

$ cut -f3,5 mydata.tsv

$ cut -d, -f2 mydata.csv| sed -e 's/"//g'
```

XML and HTML

- **Data is commonly made available in XML or HTML format**
  - However, neither is an ideal format for analysis at scale
- **XML is a self-describing hierarchical text format**
  - XML is well-formed and can be validated for compliance
  - Verbose format consumes much storage and memory
- **HTML is a closely related type used for Web pages**
  - Likelier to deviate from spec and have less structure than XML
  - Content and formatting intertwined, especially in older documents

```
$ head -n4 customers.xml
<customer>
   <id>1234</id>
   <name type="display">Smith, Jane</name>
</customer>

$ perl -ne 'print m|<id>\s*(\d+)\s*</id>(\n)|;' < customers.xml
```

JSON

- **JSON is an alternative to XML**
  - It offers many of the benefits, but with fewer drawbacks
  - Format is also hierarchical and self-describing
  - Much less verbose than XML
- **Despite its JavaScript origins, it's supported by many languages**

```
{
   "id":573698,
   "name":"John Smith",
   "address":"123 Hadoop Drive",
   "zipcode":"90210",
   "email":"jsmith@example.com",
   "phone_numbers": [
        { "type":"mobile", "number":"(213) 555-1953" },
        { "type":"work", "number":"(310) 555-2752" },
        { "type":"home", "number":"(310) 555-7419" },
    ],
}
```

Binary Input Formats

- **Not all data collected is text-based**
  - Images
  - Spreadsheets
  - Word processor and PDF documents
  - Audio and video
- **These formats are not necessarily ideal for analysis**
  - Not natively supported by Hadoop or ecosystem tools
  - Analysis typically involves format-specific custom code
- **Often better to convert to a text-based format before processing**
  - For example, convert Excel to CSV, or PDF to text
  - This may not be possible for some formats (such as images)
- **In some cases, only the metadata is actually needed for analysis**

Evaluating Input Data

- Data formats
- **Data quantity**
- Data quality

Data Quantity Considerations

- **Data quantity is a defining characteristic of data science**
- **However, preliminary analysis is often best done with less data**
  - Smaller data sets mean faster execution times
  - Faster execution times allow for more iterations
  - More iterations provides more opportunities to refine solution

Filtering

- **The goal of filtering is to limit the amount of data**
  - Include only certain records
  - Exclude only certain records
  - Isolate only those fields relevant for analysis
  - Can also combine any of these approaches
- **This can have profound impact on performance**
  - Eliminating data before it is processed will usually improve performance more than any optimization of your analysis code

```
$ gunzip -c logfile.gz | grep jsmith > only_jsmith.log

$ gunzip -c logfile.gz | grep -v 'GET /img/' > no_images.log

$ egrep '^63[0-3][0-9]{2}' clients.txt | egrep -vi 'smith|johnson' | less

$ cut -f1,3,9 mydata.tsv > mydata-3cols.txt
```

Evaluating Input Data

- Data formats
- Data quantity
- **Data quality**

Common Problems

- **Inconsistencies in case of string values**
  - CA versus ca
  - Recommendation: always convert to one case
- **Inconsistencies in date formats**
  - 12/31/2012 versus Dec. 31, 2012
  - Recommendation: always convert to one format
  - A string like 20121231 occupies less space and sorts correctly
- **Inconsistencies in times**
  - Is 12:00:00 noon or midnight? What time zone?
  - Recommendation: use a 24-hour format
  - Recommendation: use a consistent time zone, such as GMT

Data Quality Overview

- **Quality problems are inevitable in a sufficiently large data set**
- **Three main types of problem**
  - Inconsistent: correct but with minor formatting variations
  - Invalid: incorrect but conforms to expected format
  - Corrupt: doesn’t conform to expected format at all

Common Problems (cont'd)

- **Differences in how missing values are represented**
    - Does it use `NULL` or N/A or zero or spaces or an empty string?
    - Recommendation: use as few representations as possible
        - May need one for strings and another for numeric fields
- **Variations in free-form input**
    - CA versus California (might also be misspelled in various ways)
        - Recommendation: limit free-form input if possible
        - Recommendation: scan to find all variations, then normalize

```
$ cut -f5 data.tsv | sort | uniq -c | sort -rn
9887 CA
   8 California
   3 CA.
   1 Cailfornia
   1 Caleefornya
```

Identifying Bad Data

- **Small scale strategies**
  - Examine columns with UNIX tools like `head`, `cut`, and `awk`
  - Write custom code that inspects records
- **Large scale strategies**
  - Use counters in a Hadoop job to count bad records
  - Use logging in a Hadoop job to log unexpected data
    - Exercise caution with this approach!
    - Only log *bad* records, not *all* records

Resolution Techniques

- **How do you fix the bad data once you've identified it?**
  - Fix the data upstream to avoid the issue altogether
  - Pre-process the data to fix the problem before analysis
  - Correct bad data on the fly during analysis
  - Ignore bad data as you find it during analysis