ENTITY RESOLUTION

CS1951A INTRO TO DATA SCIENCE

Tim Kraska <tim_kraska@brown.edu>
ANNOUNCEMENT

1) Finding Project Team-Members

2) Bash Lab
DATA INTEGRATION

Schema Matching

Entity Resolution

deduplication, entity clustering, merge/purge, fuzzy match, record linkage, approximate match...

Data Fusion
# REAL WORLD DATA

<table>
<thead>
<tr>
<th>Customer</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Name</td>
<td>J Smith</td>
<td>Mary Jones</td>
<td>Bob Wilson</td>
<td>M Jones</td>
<td>Robert Wilson</td>
<td>James Smith</td>
</tr>
<tr>
<td>Street</td>
<td>123 University Ave</td>
<td>245 3rd St</td>
<td>345 Broadway</td>
<td>245 Third Street</td>
<td>345 Broadway St</td>
<td>123 Univ Ave</td>
</tr>
<tr>
<td>City</td>
<td>Seattle</td>
<td>Redmond</td>
<td>Seattle</td>
<td>Redmond</td>
<td>Seattle</td>
<td>Seattle</td>
</tr>
<tr>
<td>State</td>
<td>Washington</td>
<td>WA</td>
<td>Washington</td>
<td>NULL</td>
<td>WA</td>
<td>WA</td>
</tr>
<tr>
<td>P-Code</td>
<td>98106</td>
<td>98052-1234</td>
<td>98101</td>
<td>98052</td>
<td>98101</td>
<td>NULL</td>
</tr>
<tr>
<td>Age</td>
<td>42</td>
<td>30</td>
<td>19</td>
<td>299</td>
<td>19</td>
<td>41</td>
</tr>
</tbody>
</table>

- **Inconsistent representation**: The state for Mary Jones is listed as WA instead of Washington.
- **Duplicate Records**: Customer 3 and 5 have the same address and city, indicating a duplicate entry.
- **Typos**: The address for James Smith is incorrectly listed as Univ Ave instead of Univ Ave.
- **Missing Information**: The city and state for J Widom are missing.

...
REAL WORLD DATA

- How many customers do I have?

Wrong answer because of duplicate records!

- How many customers by state?

<table>
<thead>
<tr>
<th>State</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>60</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>WA</td>
<td>1200</td>
</tr>
<tr>
<td>Washington</td>
<td>50</td>
</tr>
<tr>
<td>Wasington</td>
<td>2</td>
</tr>
</tbody>
</table>

What about if you give this data to a ML algorithm?
ENTITY RESOLUTION

“[The] problem of identifying and linking/grouping different manifestations of the same real world object.”

Challenges

• Fundamental ambiguity
• Diversity in representations (format, truncation, ambiguity)
• Errors
• Missing data
• Records from different times
• Relationships in addition to equality
## Customer

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>P-Code</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>J Smith</td>
<td>123 University Ave</td>
<td>Seattle</td>
<td>Washington</td>
<td>98106</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>Mary Jones</td>
<td>245 3rd St</td>
<td>Redmond</td>
<td>WA</td>
<td>98052-1234</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>Bob Wilson</td>
<td>345 Broadway</td>
<td>Seattle</td>
<td>Washington</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>M Jones</td>
<td>245 Third Street</td>
<td>Redmond</td>
<td>NULL</td>
<td>98052</td>
<td>299</td>
</tr>
<tr>
<td>5</td>
<td>Robert Wilson</td>
<td>345 Broadway St</td>
<td>Seattle</td>
<td>WA</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>James Smith</td>
<td>123 Univ Ave</td>
<td>Seattle</td>
<td>WA</td>
<td>NULL</td>
<td>41</td>
</tr>
<tr>
<td>7</td>
<td>J Widom</td>
<td>123 University Ave</td>
<td>Palo Alto</td>
<td>CA</td>
<td>94305</td>
<td>NULL</td>
</tr>
</tbody>
</table>

... ... ... ... ... ... ...

Customer

12/02/2009 23 CSE 544: Data Cleaning
String Similarity function:

- $\text{Sim}(\text{string}, \text{string}) \rightarrow \text{numeric value}$

A “good” similarity function:

- Strings representing the same concept $\Rightarrow$ high similarity
- Strings representing different concepts $\Rightarrow$ low similarity
EDIT DISTANCE

EditDistance(s1, s2):

- Minimum number of edits to transform s1 to s2

Edit:

- Insert a character
- Delete a character
- Substitute a character

Note: EditDistance(s1, s2) = EditDistance(s2, s1)

“Distance” = opposite of similarity
EDIT DISTANCE

\textbf{EditDistance} ("Provdince", "Providence") = 2
Provdince $\longrightarrow$ Providence $\longrightarrow$ Providence

\textbf{EditDistance} ("Seattle", "Redmond") = 6
Seattle $\longrightarrow$ Reattle $\longrightarrow$ Redtle
Redmtle $\longrightarrow$ Redmole $\longrightarrow$ Redmone
$\longrightarrow$ Redmond
EDIT DISTANCE PROBLEMS

115th Waterman St., Providence, RI

EditDistance = 1

110th Waterman St., Providence, RI

Waterman Street, Providence, RI

EditDistance = 4

Waterman St, Providence, RI

Character Level vs. Word Level Similarity?
EDIT DISTANCE PROBLEMS

148th Ave NE, Redmond, WA

148th Ave NE, Redmond, WA

EditDist = 0

148th Ave NE, Redmond, WA

NE 148th Ave, Redmond, WA

EditDist = 4

Order sensitive Similarity?
JACCARD SIMILARITY

- Statistical measure
- Originally defined over sets
- String = set of words

\[ Jaccard(s_1, s_2) = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|} \]

- Range of values = [0,1]
JACCARD SIMILARITY

\[ Jaccard = \frac{4}{4 + 2} \approx 0.66 \]

148th Ave NE, Redmond, WA

140th Ave NE, Redmond, WA
JACCARD SIMILARITY

\[ Jaccard = \frac{5}{5} = 1.0 \]

148th Ave NE, Redmond, WA

NE 148th Ave, Redmond, WA
What is the Jaccard Similarity between:

iPad Two 16GB WiFi White

iPad 2nd generation 16GB Wifi White

(a) 3 / 8
(b) 4 / 11
(c) 4 / 7
CLICKER: WHICH JACCARD SIMILARITY IS WRONG

A) Microsoft Corporation \(\downarrow\) Jaccard = 1/3 \(\uparrow\) Microsoft Corp

B) Microsoft Corporation \(\downarrow\) Jaccard = 1/3 \(\uparrow\) Oracle Corporation

C) Waterman 115 St \(\downarrow\) Jaccard = 1/4 \(\uparrow\) 115 Waterman Street
WHAT CAN WE DO ABOUT?

Microsoft Corporation

Microsoft Corp

Microsoft Corporation

Oracle Corporation
JACCARD SIMILARITY

Weight Function = \( wt: Elements \rightarrow \mathbb{R}^+ \)

\[
W_{tJaccard}(s_1, s_2) = \frac{wt(s_1 \cap s_2)}{wt(s_1 \cup s_2)}
\]

\[wt(s) = \sum_{e \in s} wt(e)\]

\[wt("Microsoft") > wt("Corporation")\]
\[Wt("Oracle") > wt("Corporation")\]
• IDF: Inverse Document Frequency

\[ wt(word) = \log_e \left( \frac{\text{size of corpus}}{\text{frequency}(word)} \right) \]

• frequency(word) = defined using some “corpus”:
  • large table of records
  • Wikipedia?
**IDF WEIGHTED JACCARD**

**Microsoft Corporation**

\[
\text{WtJaccard} = \frac{12.21}{12.21 + 4.21 + 4.38} = \frac{12.21}{20.8} = 0.59
\]

**Microsoft Corp**

**Microsoft Corporation**

\[
\text{WtJaccard} = \frac{4.21}{26.57} = 0.16
\]

**Oracle Corporation**

\[
\log_e \left( \frac{1,000,000}{5} \right)
\]

<table>
<thead>
<tr>
<th>Word</th>
<th>Freq</th>
<th>IDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft</td>
<td>5</td>
<td>12.21</td>
</tr>
<tr>
<td>Oracle</td>
<td>39</td>
<td>10.15</td>
</tr>
<tr>
<td>Corporation</td>
<td>14782</td>
<td>4.21</td>
</tr>
<tr>
<td>Corp</td>
<td>12496</td>
<td>4.38</td>
</tr>
</tbody>
</table>

Corpus size = 1M records
OTHER SIMILARITY FUNCTIONS

➢ Affine edit distance
➢ Cosine similarity
➢ Hamming distance
➢ Generalized edit distance
➢ Jaro distance
➢ Monge-Elkan distance
➢ Q-gram
➢ Smith-Warerman distance
➢ Soundex distance
➢ TF/IDF
➢ …many more

• No universally good similarity function
• Choice of similarity function depends on domains of interest, data instances, etc.
# RECORD MATCHING PROBLEMS

**Customer**

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>P-Code</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>J Smith</td>
<td>123 University Ave</td>
<td>Seattle</td>
<td>Washington</td>
<td>98106</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>Mary Jones</td>
<td>245 3rd St</td>
<td>Redmond</td>
<td>WA</td>
<td>98052-1234</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>Bob Wilson</td>
<td>345 Broadway</td>
<td>Seattle</td>
<td>Washington</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>M Jones</td>
<td>245 Third Street</td>
<td>Redmond</td>
<td>NULL</td>
<td>98052</td>
<td>299</td>
</tr>
<tr>
<td>5</td>
<td>Robert Wilson</td>
<td>345 Broadway St</td>
<td>Seattle</td>
<td>WA</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>James Smith</td>
<td>123 Univ Ave</td>
<td>Seattle</td>
<td>WA</td>
<td>NULL</td>
<td>41</td>
</tr>
<tr>
<td>7</td>
<td>J Widom</td>
<td>123 University Ave</td>
<td>Palo Alto</td>
<td>CA</td>
<td>94305</td>
<td>NULL</td>
</tr>
</tbody>
</table>

---

\[
W_t \cdot c \cdot a \cdot f \Rightarrow 0.57 \quad 0.9 \cdot a \quad 1.0 \quad 0.0 \quad 1.0 \quad 1.0
\]
COMBINING SIMILARITY FUNCTIONS

Record Pair → Vector of similarity scores → Fn → Match/Non-Match

Features

Binary Classification

Jacc(Name) Jacc(Street) Edit(City) Edit(State) Edit(PostalCode) Equality(Age)
# LEARNING-BASED APPROACH

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>Zip Code</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob Wilson</td>
<td>345 Broadway</td>
<td>Seattle</td>
<td>WA</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>Robert Wilson</td>
<td>345 Broadway St</td>
<td>Seattle</td>
<td>WA</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>B Wilson</td>
<td>123 Broadway</td>
<td>Boise</td>
<td>Idaho</td>
<td>83712</td>
<td>19</td>
</tr>
<tr>
<td>Robert Wilson</td>
<td>345 Broadway St</td>
<td>Seattle</td>
<td>WA</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>M Jones</td>
<td>245 3rd St</td>
<td>Redmond</td>
<td>WA</td>
<td>98052-1234</td>
<td>30</td>
</tr>
<tr>
<td>Mary Jones</td>
<td>245 Third Street</td>
<td>Redmond</td>
<td>NULL</td>
<td>98052</td>
<td>299</td>
</tr>
<tr>
<td>Robert Wilson</td>
<td>345 Broadway St</td>
<td>Seattle</td>
<td>WA</td>
<td>98101</td>
<td>19</td>
</tr>
</tbody>
</table>

- Match
- Non-Match
LEARNING BASED APPROACH

Jaccard(Name) vs. Jaccard(Street)
LEARNING BASED APPROACH

\[ \text{Jaccard(Name)} \geq 1 \]

\[ \text{Jaccard(Street)} \geq 0.89 \]
DATA INTEGRATION

Schema Matching → Entity Resolution → Data Fusion
DATA FUSION’S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

- Resolves inconsistency across diversity of sources

Voting

Source Quality

Copy Detection

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jagadish</td>
<td>UM</td>
<td>ATT</td>
<td>UM</td>
<td>UM</td>
<td>UI</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UW</td>
<td>UW</td>
<td>UW</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>ATT</td>
<td>BEA</td>
<td>BEA</td>
<td>BEA</td>
</tr>
<tr>
<td>Franklin</td>
<td>UCB</td>
<td>UCB</td>
<td>UMD</td>
<td>UMD</td>
<td>UMD</td>
</tr>
</tbody>
</table>
Data fusion: voting + source quality + copy detection

- **Voting**
  - Jagadish: UM, ATT, UM
  - Dewitt: MSR, MSR, UW
  - Bernstein: MSR, MSR, MSR
  - Carey: UCI, ATT, BEA
  - Franklin: UCB, UCB, UMD

- **Source Quality**

- **Copy Detection**
DATA FUSION’S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

• Supports difference of opinion

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jagadish</td>
<td>UM</td>
<td>ATT</td>
<td>UM</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UW</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>ATT</td>
<td>BEA</td>
</tr>
<tr>
<td>Franklin</td>
<td>UCB</td>
<td>UCB</td>
<td>UMD</td>
</tr>
</tbody>
</table>
Data fusion: voting + source quality + copy detection

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jagadish</td>
<td>UM</td>
<td>ATT</td>
<td>UM</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UW</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>ATT</td>
<td>BEA</td>
</tr>
<tr>
<td>Franklin</td>
<td>UCB</td>
<td>UCB</td>
<td>UMD</td>
</tr>
</tbody>
</table>
**Data fusion**: voting + source quality + copy detection

- Gives more weight to knowledgeable sources

### Table

<table>
<thead>
<tr>
<th>Source</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jagadish</td>
<td>UM</td>
<td>ATT</td>
<td>UM</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UW</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>ATT</td>
<td>BEA</td>
</tr>
<tr>
<td>Franklin</td>
<td>UCB</td>
<td>UCB</td>
<td>UMD</td>
</tr>
</tbody>
</table>
DATA FUSION’S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jagadish</td>
<td>UM</td>
<td>ATT</td>
<td>UM</td>
<td>UM</td>
<td>UI</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UW</td>
<td>UW</td>
<td>UW</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>ATT</td>
<td>BEA</td>
<td>BEA</td>
<td>BEA</td>
</tr>
<tr>
<td>Franklin</td>
<td>UCB</td>
<td>UCB</td>
<td>UMD</td>
<td>UMD</td>
<td>UMD</td>
</tr>
</tbody>
</table>
DATA FUSION’S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

- Reduces weight of copier sources

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jagadish</td>
<td>UM</td>
<td>ATT</td>
<td>UM</td>
<td>UM</td>
<td>UI</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UW</td>
<td>UW</td>
<td>UW</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>ATT</td>
<td>BEA</td>
<td>BEA</td>
<td>BEA</td>
</tr>
<tr>
<td>Franklin</td>
<td>UCB</td>
<td>UCB</td>
<td>UMD</td>
<td>UMD</td>
<td>UMD</td>
</tr>
</tbody>
</table>
DATA INTEGRATION

Schema Matching

Entity Resolution

Data Fusion

Schema Alignment

Deduplication, entity clustering, merge/purge, fuzzy match, record linkage, approximate match...
SO FAR: RELATIONAL DATA

<table>
<thead>
<tr>
<th>Appearances</th>
<th>Team</th>
<th>Wins</th>
<th>Losses</th>
<th>Winning percentage</th>
<th>Season(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>[[Pittsburgh Steelers]]&lt;sup&gt;†&lt;/sup&gt;</td>
<td>2</td>
<td>2</td>
<td>.750</td>
<td>[[Super Bowl IX</td>
</tr>
<tr>
<td>5</td>
<td>[[Dallas Cowboys]]&lt;sup&gt;*&lt;/sup&gt;</td>
<td>3</td>
<td>2</td>
<td>.625</td>
<td>[[Super Bowl V</td>
</tr>
</tbody>
</table>
[Sean Kandel et al: Research directions in data wrangling: Visualizations and transformations for usable and credible data, Information Visualization, 2011]
THREE EXTREMELY POWERFUL TOOLS

1) grep
Basic syntax:

grep 'regexp' filename

or equivalently (using UNIX pipelining):

cat filename | grep 'regexp'
WHAT IS A REGULAR EXPRESSION?

A regular expression (regex) describes a set of possible input strings.

*Regular expressions* descend from a fundamental concept in Computer Science called *finite automata* theory

*Regular expressions* are endemic to Unix

- vi, ed, sed, and emacs
- awk, tcl, perl and Python
- grep, egrep, fgrep
- compilers
REGULAR EXPRESSIONS

The simplest regular expressions are a string of literal characters to match.

The string *matches* the regular expression if it contains the substring.
UNIX Tools rocks.

match

UNIX Tools sucks.

match

UNIX Tools is okay.

no match
A regular expression can match a string in more than one place.
The `. ` regular expression can be used to match any character.
OR

\( a|b^* \) denotes \( \{ \varepsilon, "a", "b", "bb", "bbb", \ldots \} \)

\((a|b)^*\) denotes the set of all strings with no symbols other than "a" and "b", including the empty string: \( \{ \varepsilon, "a", "b", "aa", "ab", "ba", "bb", "aaa", \ldots \} \)

\( ab^*(c) \) denotes the set of strings starting with "a", then zero or more "b"s and finally optionally a "c": \{"a", "ac", "ab", "abc", "abb", "abbc", \ldots \}
Character classes \([\ ]\) can be used to match any specific set of characters.
Character classes can be negated with the \[ ^\] syntax.

```
beat a brat on a boat
```

regular expression \[ b[^eo]a t \] matches the text "beat a brat on a boat".
MORE ABOUT CHARACTER CLASSES

- \([aeiou]\) will match any of the characters a, e, i, o, or u
- \([kK]orn\) will match korn or Korn

Ranges can also be specified in character classes

- \([1–9]\) is the same as \([123456789]\)
- \([abcde]\) is equivalent to \([a–e]\)
- You can also combine multiple ranges
  - \([abcde123456789]\) is equivalent to \([a–e1–9]\)
- Note that the – character has a special meaning in a character class *but only* if it is used within a range, \([-123]\) would match the characters –, 1, 2, or 3
NAMED CHARACTER CLASSES

Commonly used character classes can be referred to by name (*alpha*, *lower*, *upper*, *alnum*, *digit*, *punct*, *cntrl*)

Syntax `[:name:]`

- `[a-zA-Z]` `[:alpha:]`
- `[a-zA-Z0-9]` `[:alnum:]`
- `[45a-z]` `[:lower:]`

Important for portability across languages
Anchors are used to match at the beginning or end of a line (or both).

^ means beginning of the line

$ means end of the line
beat a brat on a boat

regular expression

\^ b [eor] a t

beat a brat on a boat

match

regular expression

b [eor] a t $

beat a brat on a boat

match

\^word$  \^$
**REPETITION**

The * is used to define zero or more occurrences of the *single* regular expression preceding it.
I got mail, \texttt{yaaaaaay!}

For me to \texttt{poop on}.
MATCH LENGTH

A match will be the longest string that satisfies the regular expression.

regular expression

Scrapple from the apple.

no

no

yes
REPETITION RANGES

Ranges can also be specified

- `{ }` notation can specify a range of repetitions for the immediately preceding regex
- `{n}` means exactly $n$ occurrences
- `{n,}` means at least $n$ occurrences
- `{n,m}` means at least $n$ occurrences but no more than $m$ occurrences

**Example:**

- `. {0,}` same as `.*`
- `a {2,}` same as `aaa*`
• grep comes from the ed (Unix text editor) search command “global regular expression print” or `g/re/p`

• This was such a useful command that it was written as a standalone utility

• There are two other variants, `egrep` and `fgrep` that comprise the `grep` family

• `grep` is the answer to the moments where you know you want the file that contains a specific phrase but you can’t remember its name
FAMILY DIFFERENCEs

grep - uses regular expressions for pattern matching

fgrep - file grep, does not use regular expressions, only matches fixed strings but can get search strings from a file

egrep - extended grep, uses a more powerful set of regular expressions but does not support backreferencing, generally the fastest member of the grep family

agrep – approximate grep; not standard
Sometimes it is handy to be able to refer to a match that was made earlier in a regex

This is done using backreferences

- \n is the backreference specifier, where \n is a number

Looks for \n\n \n\n
For example, to find if the first word of a line is the same as the last:

- ^([^[:alpha:]]{1,}) .* \1$
  - The ([[:alpha:]]{1,}) matches 1 or more letters
Dollar amount with optional cents

- \$[0-9]+(\.[0-9][0-9])?

Time of day

- (1[012]|1[1-9]):[0-5][0-9] (am|pm)

HTML headers <h1> <H1> <h2> ...

- <![hH][1-4]>
CLICKER QUESTION

Select the string for which the regular expression ‘..\19..’ would find a match:

a) “12.1000”

b) “123.1900”

c) “12.2000”

d) the regular expression does not find a match for any of the strings above
CLICKER QUESTION

Choose the pattern that finds all filenames in which

1. the first letters of the filename are chap,
2. followed by two digits,
3. followed by some additional text,
4. and ending with a file extension of .doc

For example: chap23Production.doc

a) chap[0-9]*.doc
b) chap*[0-9]doc
c) chap[0-9][0-9].*\doc
d) chap*doc
**GREP FAMILY**

Syntax

```bash
grep [-hilnv] [-e expression] [filename]
egrep [-hilnv] [-e expression] [-f filename] [expression] [filename]
fgrep [-hilnxv] [-e string] [-f filename] [string] [filename]
```

- `-h` Do not display filenames
- `-i` Ignore case
- `-l` List only filenames containing matching lines
- `-n` Precede each matching line with its line number
- `-v` Negate matches
- `-x` Match whole line only (fgrep only)
- `-e expression` Specify expression as option
- `-f filename` Take the regular expression (egrep) or a list of strings (fgrep) from filename
THREE EXTREMELY POWERFUL TOOLS

1) grep

Basic syntax:

    grep 'regexp' filename

or equivalently (using UNIX pipelining):

    cat filename | grep 'regexp'

2) sed – stream editor

Basic syntax

    sed 's/regexp/replacement/g' filename

For each line in the input, the portion of the line that matches regexp (if any) is replaced with replacement.

Sed is quite powerful within the limits of operating on single line at a time.

You can use \( \) to refer to parts of the pattern match.
THREE EXTREMELY POWERFUL TOOLS

Awk

Finally, awk is a powerful scripting language (not unlike perl). The basic syntax of awk is:

    awk -F',' 'BEGIN{commands}
       /regexp1/ {command1} /regexp2/ {command2}
       END{commands}'

- For each line, the regular expressions are matched in order, and if there is a match, the corresponding command is executed (multiple commands may be executed for the same line).
- BEGIN and END are both optional.
- The -F',' specifies that the lines should be split into fields using the separator ",", and those fields are available to the regular expressions and the commands as $1, $2, etc.
- See the manual (man awk) or online resources for further details.
grep "created_at" twitter.json
  | sed 's/.*/"user":{"id":\([0-9]*\)\}./\1/'
  | sort | uniq -c | sort -n | tail -5"
DATA WRANGLER / TRIFACTA

http://vis.stanford.edu/wrangler/app/

<table>
<thead>
<tr>
<th>abc</th>
<th>Event_ID</th>
<th>@</th>
<th>User_Email</th>
<th>Access_Date</th>
<th>Screen_Detail</th>
<th>Device_Manufacturer</th>
<th>Device_OS_Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>2594 Categories</td>
<td>2593 Categories</td>
<td>Sep 12</td>
<td>Dec 12</td>
<td>00:00</td>
<td>23:00</td>
<td>4 Categories</td>
<td>8 Categories</td>
</tr>
<tr>
<td>1 DCA1000049084</td>
<td>luctus.vulpate.misi@feliaN</td>
<td>2012-09-13</td>
<td>17:37:34</td>
<td>samsung</td>
<td>Android 4.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 DCA1000049085</td>
<td><a href="mailto:velit@Nuncpolvinar.edu">velit@Nuncpolvinar.edu</a></td>
<td>2012-10-17</td>
<td>02:43:32</td>
<td>android_name=atarget&amp;android_so=samsung</td>
<td>Windows Phone 7.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 DCA1000049086</td>
<td>nunc.risis.vartus@Nullovulpus</td>
<td>2012-11-28</td>
<td>16:43:16</td>
<td>android_name=holidaypromo2&amp;at=samsung</td>
<td>Android 4.0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 DCA1000049087</td>
<td>fermen tum.vel@turpisnecmouri</td>
<td>2012-10-15</td>
<td>05:44:38</td>
<td>android_name=atarget&amp;android_so=samsung</td>
<td>Windows Phone 7.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 DCA1000049088</td>
<td>vulputat.ornare@velitaevelinit</td>
<td>2012-10-14</td>
<td>16:32:41</td>
<td>android_name=holidaypromo1&amp;at=samsung</td>
<td>Windows Mobile 6.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 DCA1000049089</td>
<td><a href="mailto:Duis.elementum@Mauris.eu.net">Duis.elementum@Mauris.eu.net</a></td>
<td>2012-11-03</td>
<td>08:22:33</td>
<td>android_name=atarget&amp;android_so=Nokia</td>
<td>Samsung</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 DCA1000049090</td>
<td><a href="mailto:non.arcu.Vivamus@Proinisi.l">non.arcu.Vivamus@Proinisi.l</a></td>
<td>2012-10-23</td>
<td>14:56:00</td>
<td>null</td>
<td>Android 3.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 DCA1000049091</td>
<td><a href="mailto:nec@dictum.ca">nec@dictum.ca</a></td>
<td>2012-11-18</td>
<td>17:16:43</td>
<td>android_name=holidaypromo1&amp;at=Nokia</td>
<td>Windows Mobile 6.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 DCA1000049092</td>
<td><a href="mailto:Aenean@Vivamusnisi.com">Aenean@Vivamusnisi.com</a></td>
<td>2012-09-27</td>
<td>02:24:50</td>
<td>null</td>
<td>Android 4.1.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 DCA1000049093</td>
<td>in.hendrerit.consectetur@eu</td>
<td>2012-10-17</td>
<td>16:36:26</td>
<td>null</td>
<td>Windows Mobile 6.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 DCA1000049094</td>
<td>urna.Nunc@com</td>
<td>2012-10-22</td>
<td>12:49:24</td>
<td>null</td>
<td>Windows Mobile 6.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 DCA1000049095</td>
<td>faucibus.Luctus@porttitorero</td>
<td>2012-11-12</td>
<td>04:09:55</td>
<td>android_name=holidaypromo2&amp;at=null</td>
<td>Windows Mobile 6.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 DCA1000049096</td>
<td>Donec@amet</td>
<td>2012-12-19</td>
<td>10:15:33</td>
<td>null</td>
<td>Windows Mobile 6.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 DCA1000049097</td>
<td><a href="mailto:lobortis@sed.co">lobortis@sed.co</a></td>
<td>2012-12-12</td>
<td>10:16:56</td>
<td>android_name=holidaypromo1&amp;at=Nokia</td>
<td>Android 4.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 DCA1000049098</td>
<td>amet.visis.Donec@Integerinc</td>
<td>2012-12-16</td>
<td>18:28:18</td>
<td>null</td>
<td>Windows Mobile 6.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 DCA1000049099</td>
<td><a href="mailto:mollis@turpisnulla.co">mollis@turpisnulla.co</a></td>
<td>2012-10-16</td>
<td>04:17:49</td>
<td>android_name=holidaypromo2&amp;at=Nokia</td>
<td>Windows Mobile 6.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17 DCA1000049100</td>
<td><a href="mailto:orci.adipiscing.namo@nasa.co">orci.adipiscing.namo@nasa.co</a></td>
<td>2012-11-03</td>
<td>11:47:35</td>
<td>null</td>
<td>Windows Mobile 6.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18 DCA1000049101</td>
<td>blandit@Phasellusnullar fusce</td>
<td>2012-09-14</td>
<td>02:24:31</td>
<td>null</td>
<td>Windows Mobile 6.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19 DCA1000049102</td>
<td>tinciduntl adipiscing,Mauris</td>
<td>2012-10-13</td>
<td>13:46:24</td>
<td>null</td>
<td>Windows Mobile 6.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 DCA1000049103</td>
<td><a href="mailto:velit@lobortisquis.sed.net">velit@lobortisquis.sed.net</a></td>
<td>2012-11-11</td>
<td>05:06:07</td>
<td>null</td>
<td>Android 4.0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21 DCA1000049104</td>
<td><a href="mailto:Nulla.eu.neque@necomalli.co">Nulla.eu.neque@necomalli.co</a></td>
<td>2012-11-28</td>
<td>20:50:25</td>
<td>null</td>
<td>Windows Phone 7.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22 DCA1000049105</td>
<td><a href="mailto:fringilla@nullaet.org">fringilla@nullaet.org</a></td>
<td>2012-10-08</td>
<td>14:53:45</td>
<td>null</td>
<td>Windows Mobile 6.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23 DCA1000049106</td>
<td>faucibus.Luctusc@tortumenuc</td>
<td>2012-11-14</td>
<td>21:51:54</td>
<td>android_name=holidaypromo2&amp;at=Htc</td>
<td>Android 4.1.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24 DCA1000049107</td>
<td>nisl.CumDonecestmauris.com</td>
<td>2012-10-16</td>
<td>14:38:37</td>
<td>android_name=holidaypromo2&amp;at=Htc</td>
<td>Windows Mobile 6.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25 DCA1000049108</td>
<td>parturient.montes.nascetur@op</td>
<td>2012-10-23</td>
<td>04:06:42</td>
<td>android_name=atarget&amp;android_so=motorola</td>
<td>Windows Mobile 6.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26 DCA1000049109</td>
<td>nisl.Quisque@fringill@comae</td>
<td>2012-10-31</td>
<td>03:01:30</td>
<td>android_name=holidaypromo1&amp;at=samsung</td>
<td>Windows Mobile 6.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TRANSFORM EDITOR

highlight row: (date(2012, 11, 7) <= Access_Date) && (Access_Date < date(2012, 12, 27))

SUGGESTED TRANSFORMS

highlight row: date(2012, 11, 7) <= Access_Date && (Access_Date < date(2012, 12, 27))
delete row: date(2012, 11, 7) <= Access_Date && (Access_Date < date(2012, 12, 27))
keep row: (date(2012, 11, 7) <= Access_Date) && (Access_Date < date(2012, 12, 27))

SCRIPT

split rows col: column1 on: "row"
split col: column1 on: "12" header
split col: Access_Time at: 10,11
rename col: column2 to: "Access_Date"
DATA WRANGLING

A typical Trajectory

Wrangling
- Cleanse
- Merge
- Adapt

Evaluate Usability of data

Analysis
- Visualize
- Analyze

Ideal Tool combining wrangling and analysis

Visual Analytics Tools

1. Diagnose
2. Transform
3. 1st load in analysis application
4. Start of analysis
5. Repeat with new data
6. Document

Original Data Sets

Analytics Processes

Updated Data Sets

raw data

usable data

usable data + findings

Trail of data transformations

Trail of analysis = Insight Provenance

Sean Kandel et al: Research directions in data wrangling: Visualizations and transformations for usable and credible data, Information Visualization, 2011]
FACEBOOK VISUALIZATION I

[Sean Kandel et al: Research directions in data wrangling: Visualizations and transformations for usable and credible data, Information Visualization, 2011]
Sean Kandel et al: Research directions in data wrangling: Visualizations and transformations for usable and credible data, Information Visualization, 2011
[Sean Kandel et al: Research directions in data wrangling: Visualizations and transformations for usable and credible data, Information Visualization, 2011]
HOMEWORK: PANDAS

Watch: 10-minute tour of pandas
http://vimeo.com/59324550
TOKENIZATION
AND STEMMING

WORKING WITH TEXT
**TOKENIZATION**

**Input**: “Friends, Romans and Countrymen”

**Output**: Tokens

- *Friends*
- *Romans*
- *and*
- *Countrymen*

A token is an instance of a sequence of characters.

What are valid tokens?
TOKENIZATION ISSUES

Issues in tokenization:

- Finland’s capital → Finland? Finlands? Finland’s?
- Hewlett-Packard → Hewlett and Packard as two tokens?
  - state-of-the-art: break up hyphenated sequence.
  - co-education
  - lowercase, lower-case, lower case?
  - It can be effective to get the user to put in possible hyphens
- San Francisco: one token or two?
  - How do you decide it is one token?
My PGP key is 324a3df234cb23e

(800) 234-2333

• Often have embedded spaces

• Older IR systems may not index numbers
  • But often very useful: think about things like looking up error codes/stacktraces on the web
  • (One answer is using n-grams: Lecture 3)

• Will often index “meta-data” separately
  • Creation date, format, etc.
German noun compounds are not segmented

- \textit{Lebensversicherungsgesellschaftsangestellter} \rightarrow ‘life insurance company employee’
- German retrieval systems benefit greatly from a \textit{compound splitter} module (Can give a 15\% performance boost for German)

**French:** \textit{L'ensemble} \rightarrow one token or two?

- L ? L’ ? Le ?
- Want \textit{l’} ensemble to match with un ensemble (Until at least 2003, it didn’t on Google)

**Chinese and Japanese have no spaces between words:**

- 莎拉波娃现在居住在美国东南部的佛罗里达。
- Not always guaranteed a unique tokenization

**Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right**
STOP WORDS

With a stop list, you exclude from the dictionary entirely the commonest words. Intuition:

- They have little semantic content: the, a, and, to, be
- There are a lot of them: ~30% of postings for top 30 words

For building search engines, the trend is away from doing this:

- Good compression techniques means the space for including stopwords in a system is very small
- Good query optimization techniques mean you pay little at query time for including stop words.
- You need them for:
  - Phrase queries: “King of Denmark”
  - Various song titles, etc.: “Let it be”, “To be or not to be”
  - “Relational” queries: “flights to London”

In contrast for analytics: you often remove them. Why?
NORMALIZATION TO TERMS

We need to “normalize” words in indexed text as well as query words into the same form

- We want to match *U.S.A.* and *USA*

Result is terms: a **term** is a (normalized) word type, which is an entry in our IR system dictionary

We most commonly implicitly define **equivalence classes of terms** by, e.g.,

- deleting periods to form a term
  - *U.S.A., USA* $\rightarrow$ *USA*
- deleting hyphens to form a term
  - *anti-discriminatory, antidiscriminatory* $\rightarrow$ *antidiscriminatory*
NORMALIZATION: OTHER LANGUAGES

Accents: e.g., French résumé vs. resume.

Umlauts: e.g., German: Tuebingen vs. Tübingen
  • Should be equivalent

Most important criterion:
  • How are your users like to write their queries for these words?

Even in languages that standardly have accents, users often may not type them
  • Often best to normalize to a de-accented term
    • Tuebingen, Tübingen, Tubingen → Tubingen
NORMALIZATION: OTHER LANGUAGES

Normalization of things like date forms

- 7 月 30 日 vs. 7/30
- Japanese use of kana vs. Chinese characters

Tokenization and normalization may depend on the language and so is intertwined with language detection

Crucial: Need to “normalize” indexed text as well as query terms into the same form

Morgen will ich ins MIT…

Is this German “mit”?
CASE FOLDING

Reduce all letters to lower case

• exception: upper case in mid-sentence?
  • e.g., *General Motors*
  • *Fed vs. fed*
  • *Brown vs. brown*

• Often best to lower case everything, since users will use lowercase regardless of ‘correct’ capitalization…

• Again for analytics it depends

Google example:

• Query *C.A.T.*
• #1 result is for “cat” *not* Caterpillar Inc.
NORMALIZATION TO TERMS

An alternative to equivalence classing is to do asymmetric expansion

An example of where this may be useful

- Enter: *window*  Search: *window, windows*
- Enter: *windows*  Search: *Windows, windows, window*
- Enter: *Windows*  Search: *Windows*

Potentially more powerful, but less efficient
Do we handle synonyms?

- E.g., by hand-constructed equivalence classes
  - \( \text{car} = \text{automobile} \quad \text{color} = \text{colour} \)
- We can rewrite to form equivalence-class terms
  - When the document contains \( \text{automobile} \), index it under \( \text{car-automobile} \) (and vice-versa)
- Or we can expand a query
  - When the query contains \( \text{automobile} \), look under \( \text{car} \) as well

What about spelling mistakes?

- One approach is soundex, which forms equivalence classes of words based on phonetic heuristics

More in lectures 3 and 9
LEMMATIZATION

Reduce inflectional/variant forms to base form

E.g.

- *am, are, is* → *be*
- *car, cars, car's, cars'* → *car*

the boy's cars are different colors → the boy car be different color

Lemmatization implies doing “proper” reduction to dictionary headword form
STEMMING

Reduce terms to their “roots” before indexing

“Stemming” suggest crude affix chopping

• language dependent
• e.g., automate(s), automatic, automation all reduced to automat.

for example compressed and compression are both accepted as equivalent to compress.

for example compress and compress ar both accept as equvial to compress
PORTER’S ALGORITHM

Commonest algorithm for stemming English

• Results suggest it’s at least as good as other stemming options

Conventions + 5 phases of reductions

• phases applied sequentially
• each phase consists of a set of commands
• sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*
TYPICAL RULES IN PORTER

\[ sses \rightarrow ss \]
\[ ies \rightarrow i \]
\[ ational \rightarrow ate \]
\[ tional \rightarrow tion \]

Weight of word sensitive rules

\[ (m>1) \ EMENT \rightarrow \]

- replacement \rightarrow replac
- cement \rightarrow cement
OTHER STEMMERS

Other stemmers exist, e.g., Lovins stemmer

- http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm
- Single-pass, longest suffix removal (about 250 rules)

Full morphological analysis – at most modest benefits for retrieval

Do stemming and other normalizations help?

- English: very mixed results. Helps recall for some queries but harms precision on others
  - E.g., operative (dentistry) ⇒ oper
- Definitely useful for Spanish, German, Finnish, …
  - 30% performance gains for Finnish!
OTHER STEMMERS

Many of the above features embody transformations that are

- Language-specific and
- Often, application-specific

These are “plug-in” addenda to the indexing process

Both open source and commercial plug-ins are available for handling these