Data Mining

Mining the Web for Structured Data

Our view of the web so far...
- Web pages as atomic units
- Great for some applications
  - e.g., Conventional web search
- But not always the right model

Going beyond web pages
- Question answering
  - What is the height of Mt Everest?
  - Who killed Abraham Lincoln?
- Relation Extraction
  - Find all <company, CEO> pairs
- Virtual Databases
  - Answer database-like queries over web data
  - E.g., Find all software engineering jobs in Fortune 500 companies

Question Answering
- E.g., Who killed Abraham Lincoln?
- Naïve algorithm
  - Find all web pages containing the terms "killed" and "Abraham Lincoln" in close proximity
  - Extract k-grams from a small window around the terms
  - Find the most commonly occurring k-grams

Naïve algorithm works fairly well!
- Some improvements
  - Use sentence structure e.g., restrict to noun phrases only
  - Rewrite questions before matching
    - "What is the height of Mt Everest" becomes "The height of Mt Everest is <blank>"
  - The number of pages analyzed is more important than the sophistication of the NLP
- For simple questions

Reference: Dumais et al

Relation Extraction
- Find pairs (title, author)
  - Where title is the name of a book
  - E.g., (Foundation, Isaac Asimov)
- Find pairs (company, hq)
  - E.g., (Microsoft, Redmond)
- Find pairs (abbreviation, expansion)
  - (ADA, American Dental Association)
- Can also have tuples with >2 components
Relation Extraction

- Assumptions:
  - No single source contains all the tuples
  - Each tuple appears on many web pages
  - Components of tuple appear “close” together
  - Foundation, by Isaac Asimov
  - Isaac Asimov’s masterpiece, the <em>Foundation</em> trilogy
  - There are repeated patterns in the way tuples are represented on web pages

Naive approach

- Study a few websites and come up with a set of patterns e.g., regular expressions

  - letter = [A-Za-z. ]
  - title = letter(5,40)
  - author = letter(10,30)
  - <b>(title)</b> by (author)

Problems with naive approach

- A pattern that works on one web page might produce nonsense when applied to another
  - So patterns need to be page-specific, or at least site-specific
- Impossible for a human to exhaustively enumerate patterns for every relevant website
  - Will result in low coverage

Better approach (Brin)

- Exploit duality between patterns and tuples
  - Find tuples that match a set of patterns
  - Find patterns that match a lot of tuples
- DIPRE (Dual Iterative Pattern Relation Extraction)

DIPRE Algorithm

1. R ← SampleTuples
   - e.g., a small set of <title,author> pairs
2. O ← FindOccurrences(R)
   - Occurrences of tuples on web pages
   - Keep some surrounding context
3. P ← GenPatterns(O)
   - Look for patterns in the way tuples occur
   - Make sure patterns are not too general!
4. R ← MatchingTuples(P)
5. Return or go back to Step 2

Occurrences

- e.g., Titles and authors
- Restrict to cases where author and title appear in close proximity on web page

  - <li><b>Foundation</b> by Isaac Asimov (1951)
  - order = [title,author] (or [author,title])
  - denote as 0 or 1
  - prefix = "<li><b> " (limit to e.g., 10 characters)
  - middle = "</b> by "
  - suffix = "(1951)"
  - occurrence = ('Foundation','Isaac Asimov',url,order,prefix,middle,suffix)
Patterns

- **Foundation** by Isaac Asimov (1951)
- **Nightfall** by Isaac Asimov (1941)

- order = [title,author] (say 0)
- shared prefix = `<b>`
- shared middle = </b> by
- shared suffix = (19
- pattern = (order,shared prefix, shared middle, shared suffix)

URL Prefix

- Patterns may be specific to a website
  - Or even parts of it
- Add urlprefix component to pattern

<table>
<thead>
<tr>
<th>URL Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.scifi.org/bydecade/1950.html">http://www.scifi.org/bydecade/1950.html</a> occurrence:</td>
</tr>
<tr>
<td>&lt;li&gt;&lt;b&gt;Foundation&lt;/b&gt; by Isaac Asimov (1951)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>URL Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.scifi.org/bydecade/1940.html">http://www.scifi.org/bydecade/1940.html</a> occurrence:</td>
</tr>
<tr>
<td>&lt;li&gt;&lt;b&gt;Nightfall&lt;/b&gt; by Isaac Asimov (1941)</td>
</tr>
</tbody>
</table>

Example

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.scifi.org/bydecade/1950.html">http://www.scifi.org/bydecade/1950.html</a> occurrence:</td>
</tr>
<tr>
<td>Foundation, by Isaac Asimov, has been hailed...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.scifi.org/bydecade/1940.html">http://www.scifi.org/bydecade/1940.html</a> occurrence:</td>
</tr>
<tr>
<td>Nightfall, by Isaac Asimov, tells the tale of...</td>
</tr>
</tbody>
</table>

Pattern Specificity

- We want to avoid generating patterns that are too general
- One approach:
  - For pattern p, define specificity = |urlprefix|middle|prefix|suffix|
  - Suppose n(p) = number of occurrences that match the pattern p
  - Discard patterns where n(p) < n_min
  - Discard patterns p where specificity(p)n(p) < threshold
Pattern Generation Algorithm
1. Group occurrences by order and middle
2. Let \( O \) = a set of occurrences with the same order and middle
3. \( p = \text{GeneratePattern}(O) \)
4. If \( p \) meets specificity requirements, add \( p \) to set of patterns
5. Otherwise, try to split \( O \) into multiple subgroups by extending the urlprefix by one character
   - If all occurrences in \( O \) are from the same URL, we cannot extend the urlprefix, so we discard \( O \)

Extending the URL prefix
Suppose \( O \) contains occurrences from urls of the form
\[
\text{http://www.scifi.org/bydecade/195??.html}
\text{http://www.scifi.org/bydecade/194??.html}
\]

urlprefix = \text{http://www.scifi.org/bydecade/19}
When we extend the urlprefix, we split \( O \) into two subsets:
urlprefix = \text{http://www.scifi.org/bydecade/194}
urlprefix = \text{http://www.scifi.org/bydecade/195}

Finding occurrences and matches
- Finding occurrences
  - Use inverted index on web pages
  - Examine resulting pages to extract occurrences
- Finding matches
  - Use urlprefix to restrict set of pages to examine
  - Scan each page using regex constructed from pattern

Relation Drift
- Small contaminations can easily lead to huge divergences
- Need to tightly control process
- Snowball (Agichtein and Gravano)
  - Trust only tuples that match many patterns
  - Trust only patterns with high “support” and “confidence”

Pattern support
- Similar to DIPRE
- Eliminate patterns not supported by at least \( n_{\text{min}} \) known good tuples
  - either seed tuples or tuples generated in a prior iteration

Pattern Confidence
- Suppose tuple \( t \) matches pattern \( p \)
- What is the probability that tuple \( t \) is valid?
- Call this probability the confidence of pattern \( p \), denoted \( \text{conf}(p) \)
  - Assume independent of other patterns
- How can we estimate \( \text{conf}(p) \)?
Categorizing pattern matches
- Given pattern p, suppose we can partition its matching tuples into groups p.positive, p.negative, and p.unknown
- Grouping methodology is application-specific

Categorizing Matches
- e.g., Organizations and Headquarters
  - A tuple that exactly matches a known pair (org,hq) is positive
  - A tuple that matches the org of a known tuple but a different hq is negative
  - A tuple that matches a hq that is not a known city is negative
  - Assume org is key for relation
  - Assume we have a list of valid city names
  - All other occurrences are unknown

Categorizing Matches
- Books and authors
  - One possibility...
  - A tuple that matches a known tuple is positive
  - A tuple that matches the title of a known tuple but has a different author is negative
    - Assume title is key for relation
  - All other tuples are unknown
- Can come up with other schemes if we have more information
  - e.g., list of possible legal people names

Example
- Suppose we know the tuples
  - Foundation, Isaac Asimov
  - Startide Rising, David Brin
- Suppose pattern p matches
  - Foundation, Isaac Asimov
  - Startide Rising, David Brin
  - Foundation, Doubleday
  - Rendezvous with Rama, Arthur C. Clarke
- |p.positive| = 2, |p.negative| = 1, |p.unknown| = 1

Pattern Confidence (1)
- \( \text{pos}(p) = |p.\text{positive}| \)
- \( \text{neg}(p) = |p.\text{negative}| \)
- \( \text{un}(p) = |p.\text{unknown}| \)
- \( \text{conf}(p) = \frac{\text{pos}(p)}{\text{pos}(p)+\text{neg}(p)} \)

Pattern Confidence (2)
- Another definition – penalize patterns with many unknown matches
- \( \text{conf}(p) = \frac{\text{pos}(p)}{\text{pos}(p)+\text{neg}(p)+\text{un}(p)\alpha} \)
- where \( 0 < \alpha < 1 \)
Tuple confidence

- Suppose candidate tuple t matches patterns p₁ and p₂
- What is the probability that t is a valid tuple?
  - Assume matches of different patterns are independent events

Pr[t matches p₁ and t is not valid] = 1 - \text{conf}(p₁)
Pr[t matches p₂ and t is not valid] = 1 - \text{conf}(p₂)
Pr[t matches \{p₁, p₂\} and t is not valid] = (1 - \text{conf}(p₁))(1 - \text{conf}(p₂))
Pr[t matches \{p₁, p₂\} and t is valid] = 1 - (1 - \text{conf}(p₁))(1 - \text{conf}(p₂))

If tuple t matches a set of patterns P
\text{conf}(t) = 1 - \prod_{p \in P}(1 - \text{conf}(p))

Snowball algorithm

1. Start with seed set R of tuples
2. Generate set P of patterns from R
   - Compute support and confidence for each pattern in P
   - Discard patterns with low support or confidence
3. Generate new set T of tuples matching patterns P
   - Compute confidence of each tuple in T
4. Add to R the tuples t \in T with conf(t) > \text{threshold}
5. Go back to step 2

Some refinements

- Give more weight to tuples found earlier
- Approximate pattern matches
- Entity tagging

Tuple confidence

- If tuple t matches a set of patterns P
  \text{conf}(t) = 1 - \prod_{p \in P}(1 - \text{conf}(p))

- Suppose we allow tuples that don’t exactly match patterns but only approximately
  \text{conf}(t) = 1 - \prod_{p \in P}(1 - \text{conf}(p) \text{match}(t,p))