CS345A
Data Mining

Mining the Web for Structured Data

Anand Rajaraman

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

- 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

Naïve 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 naïve 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. \( R \leftarrow \text{SampleTuples} \)
   - e.g., a small set of `<title,author>` pairs
2. \( O \leftarrow \text{FindOccurrences}(R) \)
   - Occurrences of tuples on web pages
   - Keep some surrounding context
3. \( P \leftarrow \text{GenPatterns}(O) \)
   - Look for patterns in the way tuples occur
   - Make sure patterns are not too general!
4. \( R \leftarrow \text{MatchingTuples}(P) \)
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

- `<b>`Foundation`</b>` by Isaac Asimov (1951)
  - `order` = `[title,author]` (or `[author,title]`)
  - `prefix` = `<b>` `<b>`
  - `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 = 
- shared middle = 
- 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

Example

- http://www.scifi.org/bydecade/1950.html occurrence:
  - Foundation, by Isaac Asimov, has been hailed...
- http://www.scifi.org/bydecade/1940.html occurrence:
  - Nightfall, by Isaac Asimov, tells the tale of...

- order = [title, author]
- middle = 
- urlprefix = http://www.scifi.org/bydecade/19
- prefix = 
- suffix = (19)

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) < \( n_{\text{min}} \)
  - Discard patterns p where specificity(p)n(p) < threshold
Pattern Generation Algorithm

1. Group occurrences by order and middle
2. Let $O = \text{a set of occurrences with the same order and middle}$
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
- http://www.scifi.org/bydecade/194?.html

urlprefix = http://www.scifi.org/bydecade/19

When we extend the urlprefix, we split $O$ into two subsets:
- urlprefix = http://www.scifi.org/bydecade/194
- urlprefix = 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.
  - Assume org is key for relation.
  - A tuple that matches a hq that is not a known city is negative.
  - 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
- \(|p\.\text{positive}| = 2, |p\.\text{negative}| = 1, |p\.\text{unknown}| = 1\)

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 \(0 \leq \alpha \leq 1\)
Tuple confidence

- Suppose candidate tuple t matches patterns $p_1$ and $p_2$
- What is the probability that t is a valid tuple?
  - Assume matches of different patterns are independent events

\[\text{Pr}[t \text{ matches } p_1 \text{ and } t \text{ is not valid}] = 1 - \text{conf}(p_1)\]
\[\text{Pr}[t \text{ matches } p_2 \text{ and } t \text{ is not valid}] = 1 - \text{conf}(p_2)\]
\[\text{Pr}[t \text{ matches } \{p_1, p_2\} \text{ and } t \text{ is not valid}] = (1 - \text{conf}(p_1))(1 - \text{conf}(p_2))\]
\[\text{Pr}[t \text{ matches } \{p_1, p_2\} \text{ and } t \text{ is valid}] = 1 - (1 - \text{conf}(p_1))(1 - \text{conf}(p_2))\]

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 \(\text{conf}(t) > \text{threshold}\).
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
\[
\text{conf}(t) = 1 - \prod_{p \in P}(1 - \text{conf}(p) \cdot \text{match}(t, p))
\]