6 Mining the Web

Outline:

1. *Dynamic itemset counting*: Searching for *interesting* sets of items in a space too large ever to consider even each pair of items.

2. *"Books and authors"*: Sergey Brin's intriguing experiment to mine the Web for relational data.

6.1 Finding Unusual Itemsets

The problem is to find sets of words that appear together "unusually often" on the Web, e.g., "New" and "York" or {"Dutchess", "of", "York"}.

- "Unusually often" can be defined in various ways, in order to capture the idea that the number of Web documents containing the set of words is much greater than what one would expect if words were sprinkled at random, each word with its own probability of occurrence in a document.

- One appropriate way is *entropy per word in the set*. Formally, the *interest* of a set of words S is

$$\frac{\log_2\left(\frac{prob(S)}{\prod_{w\ in\ S} prob(w)}\right)}{|S|}$$

  Note that we divide by the size of S to avoid the "Bonferroni effect," where there are so many sets of a given size that some, by chance alone, will appear to be correlated.

- Example: If words a, b, and c each appear in 1% of all documents, and $S = \{a, b, c\}$ appears in 0.1% of documents, then the interestingness of S is $\big(\log_2(.001/(.01 \times .01 \times .01))\big)/3 = \log_2(1000)/3$ or about 3.3.

- Technical problem: interest is not monotone, or "downwards closed," the way high support is. That is, we can have a set S with a high value of interest, yet some, or even all, of its immediate proper subsets are not interesting. In contrast, if S has high support, then all of its subsets have support at least as high.

- Technical problem: With more than 10^8 different words appearing in the Web, it is not possible even to consider all pairs of words.

6.2 The DICE Engine

DICE (dynamic itemset counting engine) repeatedly visits the pages of the Web, in a round-robin fashion. At all times, it is counting occurrences of certain sets of words, and of the individual words in that set. The number of sets being counted is small enough that the counts fit in main memory.

From time to time, say every 5000 pages, DICE reconsiders the sets that it is counting. It throws away those sets that have the lowest interest, and replaces them with other sets.

The choice of new sets is based on the *heavy edge property*, which is an experimentally justified observation that those words that appear in a high-interest set are more likely than others to appear in other high-interest sets. Thus, when selecting new sets to start counting, DICE is biased in favor of words that already appear in high-interest sets. However, it does not rely on those words exclusively, or else it could never find high-interests sets composed of the many words it has never looked at. Some (but not all) of the constructions that DICE uses to create new sets are:

1. Two random words. This is the only rule that is independent of the heavy edge assumption, and helps new words get into the pool.

2. A word in one of the interesting sets and one random word.

3. Two words from two different interesting pairs.
4. The union of two interesting sets whose intersection is of size 2 or more.
5. $\{a, b, c\}$ if all of $\{a, b\}$, $\{a, c\}$, and $\{b, c\}$ are found to be interesting.

Of course, there are generally too many options to do all of the above in all possible ways, so a random selection among options, giving some choices to each of the rules, is used.

6.3 Books and Authors

The general idea is to search the Web for facts of a given type, typically what might form the tuples of a relation such as $Books(title, author)$. The computation is suggested by Fig. 13.

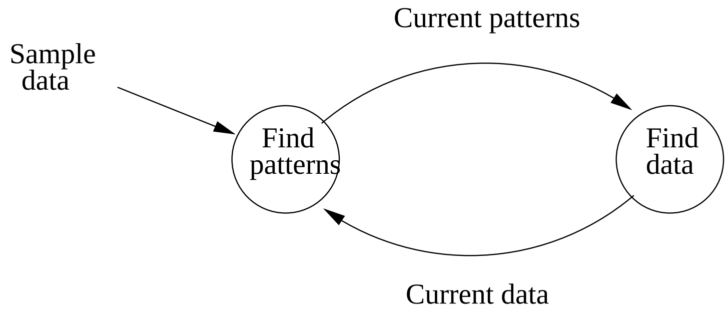

Figure 13: Extracting relations from the Web

1. Start with a sample of the tuples one would like to find. In the example discussed in the Brin paper, five examples of book titles and their authors were used.
2. Given a set of known examples, find where that data appears on the Web. If a pattern is found that identifies several examples of known tuples, and is sufficiently specific that it is unlikely to identify too much, then accept this pattern.
3. Given a set of accepted patterns, find the data that appears in these patterns, add it to the set of known data.
4. Repeat steps (2) and (3) several times. In the example cited, four rounds were used, leading to 15,000 tuples; about 95% were true title-author pairs.

6.4 What is a Pattern?

The notion suggested consists of five elements:

1. The *order*; i.e., whether the title appears prior to the author in hte text, or vice-versa. In a more general case, where tuples have more than 2 components, the order would be the permutation of components.
2. The *URL prefix*.
3. The *prefix* of text, just prior to the first of the title or author.
4. The *middle*: text appearing between the two data elements.
5. The *suffix* of text following the second of the two data elements. Both the prefix and suffix were limited to 10 characters.

Example 6.1 : A possible pattern might consist of the following:

1. Order: title then author.
2. URL prefix: `www.stanford.edu/class/`
3. Prefix, middle, and suffix of the following form:

   ```
   <LI><I>title</I> by author<P>
   ```

   Here the prefix is `<LI><I>`, the middle is `</I> by ` (including the blank after "by"), and the suffix is `<P>`. The title is whatever appears between the prefix and middle; the author is whatever appears between the middle and suffix.

The intuition behind why this pattern might be good is that there are probably lots of reading lists among the class pages at Stanford. □

To focus on patterns that are likely to be accurate, Brin used several constraints on patterns, as follows:

- Let the *specificity* of a pattern be the product of the lengths of the prefix, middle, suffix, and URL prefix. Roughly, the specificity measures how likely we are to find the pattern; the higher the specificity, the fewer occurrences we expect.
- Then a pattern must meet two conditions to be accepted:
  1. There must be at least 2 known data items that appear in this pattern.
  2. The product of the specificity of the pattern and the number of occurrences of data items in the pattern must exceed a certain threshold T (not specified).

6.5 Data Occurrences

An occurrence of a tuple is associated with a pattern in which it occurs; i.e., the same title and author might appear in several different patterns. Thus, a data occurrence consists of:

1. The particular title and author.
2. The complete URL, not just the prefix as for a pattern.
3. The order, prefix, middle, and suffix of the pattern in which the title and author occurred.

6.6 Finding Data Occurrences Given Data

If we have some known title-author pairs, our first step in finding new patterns is to search the Web to see where these titles and authors occur. We assume that there is an index of the Web, so given a word, we can find (pointers to) all the pages containing that word. The method used is essentially a-priori:

1. Find (pointers to) all those pages containing any known author. Since author names generally consist of 2 words, use the index for each first name and last name, and check that the occurrences are consecutive in the document.
2. Find (pointers to) all those pages containing any known title. Start by finding pages with each word of a title, and then checking that the words appear in order on the page.
3. Intersect the sets of pages that have an author and a title on them. Only these pages need to be searched to find the patterns in which a known title-author pair is found. For the prefix and suffix, take the 10 surrounding characters, or fewer if there are not as many as 10.

6.7 Building Patterns from Data Occurrences

1. Group the data occurrences according to their order and middle. For example, one group in the "group-by" might correspond to the order "title-then-author" and the middle "`</I> by `.
2. For each group, find the longest common prefix, suffix, and URL prefix.
3. If the specificity test for this pattern is met, then accept the pattern.
4. If the specificity test is *not* met, then try to split the group into two by extending the length of the URL prefix by one character, and repeat from step (2). If it is impossible to split the group (because there is only one URL) then we fail to produce a pattern from the group.

Example 6.2: Suppose our group contains the three URL's:

```
www.stanford.edu/class/cs345/index.html
www.stanford.edu/class/cs145/intro.html
www.stanford.edu/class/cs140/readings.html
```

The common prefix is `www.stanford.edu/class/cs` . If we have to split the group, then the next character, 3 versus 1, breaks the group into two, with those data occurrences in the first page (there could be many such occurrences) going into one group, and those occurrences on the other two pages going into another. □

6.8 Finding Occurrences Given Patterns

1. Find all URL's that match the URL prefix in at least one pattern.
2. For each of those pages, scan the text using a regular expression built from the pattern's prefix, middle, and suffix.
3. Extract from each match the title and author, according the order specified in the pattern.