An Application of the Apriori Algorithm to World Wide Web Data.

Roger G. Doss

1 Introduction

The following article describes an approach to use the Apriori Algorithm on the web by applying the algorithm to text obtained from web pages. The web pages to be used will start by a user supplied initial URL, U, and a user supplied number of web pages to visit, N. The system will then consider the web site as a graph, whose root node starts at U, and then will traverse this graph searching for N adjacent nodes, which are web pages that can be reachable from the initial URL. If there are less than N adjacent web pages, the search stops.

A web page is retrieved using a HTTP client. As a page is downloaded it is parsed where all HTML tags and comments are removed except for hypertext references which are used to determine the webpages that are to be visited next and web page titles which may be stored for subsequent retrieval. The text of the web page is important, and it can be processed by using an initial word list that contains a set of keywords relevant to the data mining operation being carried out. In this scenario, the data cleaning operation would omit any words not in this set of keywords. Alternatively, one may clean the data by explicitly removing stop words, performing word stemming, and case folding. This later approach is somewhat harder, however, consider running experiments where we wish to draw some relationship between meaningful text found on web pages to the probability that a word or set of words appears with a defined support or confidence level. Do words that have, for example, a support of 90% and confidence level of 90% actually have some relevance to the purpose of the web page ? It is one thing to search for words we are interested in, but what is the intended usage of the web page ? And, can we retrieve that web page for the specific usage based on a keyword or phrase more accurately if we apply or create a probabilistic model that accurately reflects the order of appearance of words indicative of meanings and themes that run in human authored text ?

The process of obtaining a web page, starting from the initial URL, U, extracting the information relavant to our application is repeated for all adjacent web pages until N web pages have been visited, or until there are no more adjacent web pages. Each page is treated as a transaction which maybe stored as a line in a text file whose format may resemble the following:

```
TRANSACTION:=URL DELIM [TITLE DELIM] DELIM (KEYWORD)+
URL:=http://hostname|ip address[:port]/path[;parameters][?query]
KEYWORD:=TEXT|TEXT WDELIM
TITLE:=<title>TEXT</title>
TEXT:={a-z|0-9|_}+
DELIM:=|
WDELIM:=,
```

As mentioned above, the actual keywords can be all meaningful words on the respective page or a member of the set of keywords specified by a user. If one was to examine all meaningful words, it may be approperiate to introduce a database to store this information in order to avoid performing associations on words that are not relavant to our immediate goals. A subsequent query would then produce a subset of words that match the desired criteria.

Once a transaction file has been created, the Apriori algorithm can be applied to this data and Association Rule data mining can be performed. The user is expected

to provide the desired confidence and support levels. A database can be used to store the computed support and confidence levels as well as the rules. This database may allow subsequent queries to be peformed where one may retreive information without explicitly having to go through the process of downloading the web pages and parsing out their contents. Alternatively, the system can just output the results directly to the user; for example, it can be returned via a command line interface or a CGI script.

2 Example System Design

2.1 HTTP Client

Given a URL, establish a TCP connection with the specified site and issue an HTTP GET request for the desired web page. The web page may be stored in a temporary file or in memory for subsequent processing.

The following illustrates an HTTP client algorithm:

```
socket
gethostbyname { For example, www.redhat.com. }
connect
writeout { GET / HTTP/1.0\r\n\r\n }
while(true) {
      select { Wait for IO to occur. }
      readin { One may read for a set buffer size, say 4096.
               When the read returns the number of bytes less than
               the block size, we need to NULL terminate the buffer.
      }
      if (no bytes read) {
            break;
      }
}
```

An example HTTP response may be:

```
HTTP/1.1 200 OK
Date: Sat, 03 Nov 2001 22:21:52 GMT
Server: Apache/1.3.19 (Unix)  (Red-Hat/Linux) mod_ssl/2.8.1 OpenSSL/0.9.5a
Set-Cookie: Apache=24.189.85.61.299551004826112660; path=/; expires=Thu, 02-Nov-06 22:21:52 GMT
Connection: close
Content-Type: text/html

<html>
This is a simplified HTML example, though the header is not.
</html>
```

The first line of any HTTP server response will indicate the protocol that it is implementing. In this case, although the client is stating that it is providing a HTTP version 1.0 request, it can accept a 1.1 response for several reasons. Most importantly is that we are not interested in the protocol per sae, only the data that is being provided to us, therefore, the headers are used minimally. In fact, we only care for the first line indicating a status of "200 OK", otherwise we ignore the response and the "Content-Type:". The latter is used so that the system can assert that it is dealing with a text/html page.

One should also note that the URL must be broken into its subcomponents in order for it to be used. The protocol portion of the URL only indicates that the resource can be reached via the specified protocol, and thus allows one to ignore other protocols that are not of interest. The hostname—ipaddress is used by gethostbyname/gethostbyaddress which in turn is resolved by the Domain Name System as approperiate. Similarly, the port number is used to configure the socket, usually port 80[1] The location portion of the URL is what is requested as the second argument to the GET. The carriage return/line feed sequence indicates to the corresponding server that this is the end of client transmission.

2.2 Data Cleaning

Given a web page, extract meaningul text from it. This is where much of the work can be spent due to the nature of user supplied, "real-world" text data. The structure of this module should ideally resemble a lexical analyizer[2] that scans the data and perform the approperiate operations. There are several ways to proceed. If one wishes to match words with keywords from a user supplied list, then one can view this as taking the intersection of the set of all words in the web page, with the set of keywords the user is interested in.

Alternatively, we can perform a full blown data cleaning operation, and extract all keywords from the text[3].

In both cases, portions of HTML that are not relevant, ie, not text, should be ommitted and those portions of HTML that are relavant, such as titles and hypertext references need to be extracted and further processed. The latter of which can be done in a seperate phase, though it is more efficient to extract the hyper-text references while cleaning the data.

The following illustrates an example algorithm for processing HTML:

```
read in data as a stream
while (there are more characters in the stream) {
    if( char is not alpha numeric or one of <,>,whitespace,&,newline )
         continue;
    if( char is '&' ) {
         skip HTML escape codes.
    }
    if( char is alpha ) {
         record in buffer.
    }
    if( char is whitespace ) {
         remove alpha characters from buffer and store in a set.
    }
    if( char is '<' ) {
         read till '>'.
         These are HTML markup. Find the (href="http://") sequence
         and obtain an adjacent web page, store it in it a location
         for processing.
         Record a title if one exists.
```

[1] Handling SSL is not necessary.

[2] It may be more approperiate to use LEXX to parse the data as it would simplify this process by reducing programmer error in terms of parsing textual data.

[3] This was mentioned earlier, it is more useful in the case were one is looking for associations solely based on support and confidence levels. Perhaps also in experimenting with what support and confidence levels reflect words that are indicative a given web page's usage and/or meaning.

```
        Ignore all other HTML tags.
    }
}

Perform set intersection operation with the set of keywords
the user is interested in. Store the results in the transaction file.
```

2.3 HTTP Mini-Crawler

This module reads a queue of adjacent web pages obtained from the initial web page visit and visits them in turn by calling the http client to obtain the contents, performing data cleaning to obtain the text and building a transactions file reflecting information obtained from the web site. The web page retrieval stops when N web pages have been retrieved or until there are no more web pages to be retrieved. We can view this as follows:

```
Initial user output:(URL, N, confidence, support)

Mini-Crawler(URL,N) {
                url=URL
                while(N--) {
                    HTTP Client(url) {
                              Obtain Web Page and store it locally.
                    }
                    Data Cleaning(web page) {
                              Obtain important text from the web page and store locally
                              in a transactions file.
                    }
                    Obtain Nieghboring Web Pages(web page) {
                              Store adjacent web pages in a queue.
                    }
                    url=next adjacent url.
                }
                Apriori(Confidence,Support) {
                       Use the transactions file and compute the associations.
                       Return results to user.
                }
}
```

The actual traversal can be made independent of the data cleaning and application of Apriori for future usage with different algorithms and purposes. Further, one can distribute the web crawling tasks by starting with N initial URLS and running them in parallel. While this case is not necessary to the task of applying Apriori to web text, it could be a future direction. Given a shared queue of discovered web pages, the distributed crawling engines could coordinate which URLS to visit.

2.4 Apriori

The final component to this system is to apply Apriori Association based mining to the transactions file produced. This application will produce approperiate rules based on the user supplied confidence and support levels. The only additions to the algorithm would be to handle the file format used in coordination with the data cleaning step,

and to handle text strings as opposed to simply characters or numbers. The Apriori algorithm can be stated as follows:

```
- Parse the text file to produce a set of words and their support levels.
- Reduce this set to those words with support levels equal to or greator
  than user desired support level.  Call this set C.
- We have a finite number of elements, N, in C; this is the maximum
  length pattern that can occur.  Create N sets for each iteration, I[N],
  where words, ie, strings, are the elements.
- Initialize I[0] with the data from the set C.
- for (i=1; i <= N; i++ ) {
        For each element e in set I[i-1], that is, each element from
        the previous set where:
        I[i=0], set whose elements are singular, I[i=1], set whose elements are patterns size two,
        I[i=N], set whose elements are patterns of size N.

        Let k be the set difference of e and C such that
        k is a subset with all elements in C not in e.

        For each element in k, take the union with e and
        call each result pattern Pi.  Store the pattern in a table.

        ( For example, given I[0]={a,b,c} and C={a,b,c}, starting with
          the first element in I[0], a, we take the set difference to produce
          a set k={b,c} and then produce two patterns, P0={a,b} and P1={a,c}
          by taking the union of elements in k with a. )

        For each pattern in the table, scan the database to calculate
        support.  If we view the database as a collection of M transactions,
        with each one being a set of elements, and each pattern as
        a set of elements, then support for pattern i can be determined
        by counting the number of matches in the database divided by the total
        number of transactions where a match can be computed by taking the set intersection
        of the pattern and the transaction sets.  If all elements in the residule
        set are those in the pattern, then the transaction contains this pattern.
        We can view this as, if((Pi intersect Mj) == Pi) { pattern Pi occurs in transaction Mj. }

        Compute the support for Pi as the total number of matches divided by the
        total number of transactions.  If support for Pi is less than the user desired
        support, the pattern should be discarded.  Store in I[i] all patterns
        that have the desired support.
  }
- Given I contains N sets with desired support level, compute confidence:
- For all N sets in I
     For all elements e in set I[i]
          For all elements E in set I[i-1]
               Let x be the intersection of e and E.
               Compute confidence as support for e divided by support
               for x.
               If confidence level is greator than or equal to the desired
               confidence level, store the rule as e implies x.
```

```
( For example, given I[0]={a,b,c} and I[1]={{a,b},{b,c}},
  taking the intersection of a and a,b produces a.
  Compute confidence, c, as support(a,b) / support (a).
  If this is greator than or equal to the user desired
  confidence, then store result as a rule, a,b imples a
  with confidence c. )
```

3 Conclusion

The system as described, provides a way to produce and view Association rules from a given web site. This application of Apriori to web data is important as it can be used as a valid non-trivial introduction to web mining. And, although the modules are simple enough to be implemented in a semester, they can all be extended and enhanced further with the possibility of research areas being incorporated. Several suggestions are finding better ways to traverse the web, designing databases to model data such as urls, web page titles, keywords, and support and confidence levels, building a more robust data cleaning module, perhaps one capable of understanding all of HTML and possibly even XML utilizing LEXX/YACC, applying or creating new algorithms for performing data mining on web data, and so on.