A Client-Side Web Agent for Document Categorization

D. Boley, M. Gini, K. Hastings, B. Mobasher, and J. Moore

Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455, USA
{boley,gini,hastings,mobasher,jmoore}@cs.umn.edu

Abstract

We propose a client-side agent for exploring and categorizing documents on the World Wide Web. As the user browses the Web using a usual web browser, this agent is designed to aid the user by classifying the documents the user finds most interesting into clusters. The agent carries out the task completely automatically and autonomously, with as little user intervention as the user desires. The principal novel components in this agent that make it possible are (i) a scalable hierarchical clustering algorithm and (ii) a taxonomic label generator. In this paper, we describe the overall architecture of this agent and discuss the details of the algorithms within its key components.

Keywords: Web agent, unsupervised clustering, text classification, proxy server.

1 Introduction

Many intelligent software agents have used clustering techniques in order to retrieve, filter, and categorize documents available on the World Wide Web. Clustering is also useful in extracting salient features of related Web documents to automatically formulate queries and search for other similar documents on the Web. Traditional clustering algorithms either use apriori knowledge of document structures to define a distance or similarity among these documents, or use probabilistic techniques such as Bayesian classification. Many of these traditional algorithms, however, break down as the size of the document space, and hence, the dimensionality of the corresponding feature space increases.

We propose a client-side agent for exploring and categorizing documents on the World Wide Web. As the user browses the Web using a usual web browser, this agent is designed to aid the user by

1. automatically classifying the documents retrieved by the user into clusters,

2. creating distinctive labels for each cluster which provide a summary of the documents found (much shorter than listing individual document titles),

3. searching the Web for more documents that are related to those found by the user.

The agent carries out these tasks automatically and autonomously, with as little user intervention as the user desires.

The principal novel components in this agent that make it possible are a scalable hierarchical clustering algorithm, called Principal Direction Divisive Partitioning (PDDP), and a taxonomic

label generator. The latter can lead to a query generator to retrieve new documents in autonomous fashion.

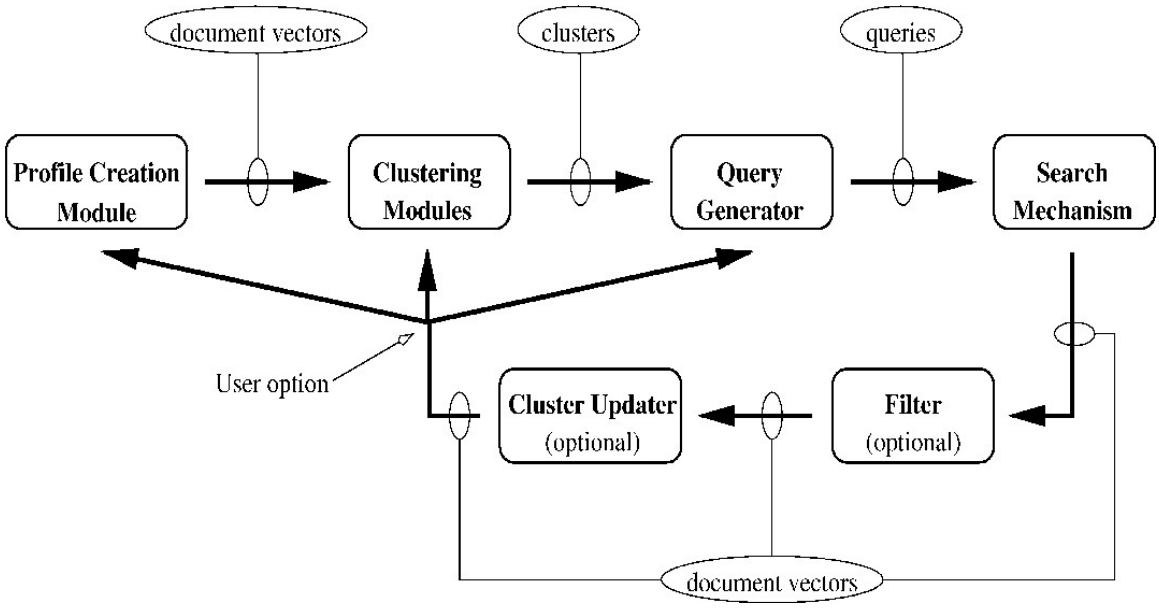

Figure 1: WebACE Architecture

In this paper, we describe the overall architecture of WebACE, an agent for document categorization and exploration that operates on Web documents, and discuss the details of the algorithms within its key components.

2 WebACE Architecture

The WebACE agent is designed to build a profile of documents which are clustered and categorized. The profile is then used to present the clusters to the user and/or to seek further related documents. The architecture of WebACE is shown in Figure 1.

As the user browses the Web, the profile creation module builds a custom profile by recording documents of interest to the user. The number of times a user visits a document and the total amount of time a user spends viewing a document are just a few methods for determining user interest (Ackerman et al., 1997, Armstrong et al., 1995, Balabanovic, Shoham, & Yun, 1995). Once WebACE has recorded a sufficient number of interesting documents, each document is reduced to a document vector and the document vectors are passed to the clustering modules. WebACE uses novel algorithms for clustering which can provide significant improvement in both run-time performance and cluster quality over traditional algorithms. These are described in Section 3.

After WebACE has found document clusters, they can be displayed to the user. WebACE also uses the clusters to generate queries and search for similar documents. WebACE submits the queries to the search mechanism and gathers the documents returned by the searches, which are in turn reduced to document vectors. This part of WebACE is described in detail in (Han et al., 1998a). These new documents can be used in a variety of ways. One option is for WebACE to

cluster the new documents, filtering out the less relevant ones. Another is to update the existing clusters by having WebACE insert the new documents into the existing clusters. Yet another is to completely re-cluster both the new and old documents.

WebACE is implemented as a browser independent Java application. Monitoring the user's browsing behavior is accomplished via a proxy server. The proxy server allows WebACE to inspect the browser's HTTP requests and the resulting responses. Upon execution, WebACE spawns a browser and starts a thread to listen for HTTP requests from the browser. As the browser makes requests, WebACE creates request threads to handle them. This allows multi-threaded browsers the capability of having multiple requests pending at one time. The lifespan of these request threads is short, i.e. the duration of one HTTP request, Conversely, the browser listener thread persists for the duration of the application. More details on the design and implementation are presented in Section 4.

3 Clustering Methods

Clustering is a discovery process that groups a set of data such that the intracluster similarity is maximized and the intercluster similarity is minimized. Clustering of documents can be done using methods that have been studied in several areas including statistics (Dubes & Jain, 1980, Lee, 1981, Cheeseman & Stutz, 1996), machine learning (Shavlik & Dietterich, 1990, Fisher, 1995), and data mining (Ng & Han, 1994, Cheeseman & Stutz, 1996).

Most of the existing approaches to document clustering are based on either probabilistic methods, or distance and similarity measures (Frakes & Baeza-Yates, 1992). Distance-based methods such as k-means analysis, hierarchical clustering (Jain & Dubes, 1988) and nearest-neighbor clustering (Lu & Fu, 1978) use a selected set of words (features) appearing in different documents as the dimensions. Each such feature vector, representing a document, can be viewed as a point in this multi-dimensional space.

There are a number of problems with clustering in this multi-dimensional space using traditional distance- or probability-based methods. We have found that hierarchical agglomeration clustering (HAC) (Duda & Hart, 1973), based on distances between sample cluster means, does a poor job on our examples. Most distance-based schemes, such as k-means analysis, do not perform very well if the dimension of the space is very large. These schemes generally require the calculation of the mean of document clusters. If the dimensionality is high, then the calculated mean values do not differ significantly from one cluster to the other. Hence the clustering based on these mean values does not produce very good clusters. It is possible to reduce the dimensionality by selecting only k-most frequent words from each document, or to use some other method to extract the salient features of each document. However, the number of features collected using these methods still tends to be very large.

Similarly, probabilistic methods such as Bayesian classification used in AutoClass (Cheeseman & Stutz, 1996, Titterington, Smith, & Makov, 1985) do not perform well when the size of the feature space is much larger than the size of the sample set or may depend on the independence of the underlying features. Web documents suffer from both high dimensionality and high correlation among the feature values. We have found AutoClass has performed poorly on our examples (Han et al., 1998a).

3.1 The PDDP Clustering Algorithm

Our clustering algorithm, Principal Direction Divisive Partitioning (PDDP) (Boley, 1997), is designed to efficiently handle very high dimensional spaces.

In the PDDP algorithm, each document is represented by a feature vector of word frequencies, scaled to unit length. The vectors are assembled into a single term frequency matrix.

The algorithm is a divisive method in the sense that it begins with all the documents in a single large cluster, and proceeds by splitting it into subclusters in recursive fashion. At each stage in the process, the method (a) selects an unsplit cluster to split, and (b) splits that cluster into two subclusters.

For part (a) we use a scatter value, measuring the average distance from the documents in a cluster to the cluster mean (Duda & Hart, 1973), though we could also use just the cluster size if it were desired to keep the resulting clusters all approximately the same size.

For part (b) we construct a linear discriminant function based on the principal direction (the direction of maximal variance). Specifically, we compute the mean of the documents within the cluster, and then the principal direction with respect to that mean. This defines a hyperplane normal to the principal direction and passing through the mean. This hyperplane is then used to split the cluster into two parts which become the two children clusters to the given cluster. This yields a splitting which is totally unsupervised.

This entire cycle is repeated as many times as desired resulting in a binary tree hierarchy of clusters in which the root is the entire document set, and each interior node has been split into two children. The leaf nodes then constitute a partitioning of the entire document set.

The cycle is terminated using on a stopping test based on the scatter value. As the algorithm proceeds, the largest scatter value among the leaf clusters is a non-increasing quantity. We also form a pseudo-cluster consisting of the centroid vectors extracted from all the leaf nodes. The scatter of this pseudo-cluster is a non-decreasing value measuring the separation between different clusters. When this latter scatter value exceeds the largest scatter value among the leaf clusters, the algorithm terminates. The user can adjust the fineness of the resulting clusters by scaling the pseudo-cluster scatter values.

The definition of the hyperplane is based on principal component analysis, similar to the Hotelling or Karhunen-Loeve Transformation (Duda & Hart, 1973). We compute the principal direction as the leading eigenvector of the sample covariance matrix. Computationally, this is the most expensive part, for which we use a fast Lanczos-based singular value solver (Golub & Van Loan, 1996). By taking advantage of the high degree of sparsity in the term frequency matrix, the Lanczos-based solver is very efficient, with cost proportional to the number of nonzeroes in the term frequency matrix. This has been discussed in more detail in (Boley, 1997).

This method differs from that of Latent Semantic Indexing (LSI) (Berry, Dumais, & O'Brien, 1995) in many ways. First of all, LSI was originally formulated for a different purpose, namely as a method to reduce the dimensionality of the search space for the purpose of handling queries, i.e. retrieving some documents given a set of search terms. Secondly, it operates on the unscaled vectors, whereas we scale the document vectors to have unit length. Thirdly, in LSI, the singular value decomposition of the matrix of document vectors itself are computed, whereas we shift the documents so that their mean is at the origin in order to compute the covariance matrix. Fourthly,

the LSI method must compute many singular vectors of the entire matrix of document vectors, perhaps on the order of 100 such singular vectors, but it must do so only once at the beginning of the processing. In our method, we must compute only the single leading singular vector (the vector u), which is considerably easier to obtain. Of course we must repeat this computation on each cluster found during the course of the algorithm, but all the later clusters are much smaller than the initial "root" cluster, and hence the later computations are much faster.

In most of our experiments, we have used the norm scaling, in which each document is represented by a feature vector of word counts, scaled to unit length in the usual Euclidean norm. This leaves the sparsity pattern untouched. An alternate scaling is the TFIDF scaling (Salton & McGill, 1983), but this scaling fills in all zero entries with nonzeroes, drastically increasing the cost of the overall algorithm by as much as a factor of 20 or more. In spite of the increased costs, TFIDF scaling did not lead to any noticeable improvement in the PDDP results in our experiments (Boley, 1998).

3.2 Performance on Large Datasets

The PDDP method has been applied successfully to various data sets, containing documents on different topics, of different styles (navigational pages, content pages, index pages. glossary pages. etc.), and with different numbers of documents. In all the experiments the pages were downloaded, labeled, and archived. This ensures a stable data sample since many pages are fairly dynamic in content.

An initial set of experiments was done with 98 documents in four broad categories (business and finance, electronic communication and networking, labor, and manufacturing). These documents were obtained in part from the Network for Excellence in Manufacturing website, on line at http://web.miep.org:80/miep/index.html. Most of these were used originally for the experiments described in (Wulfekuhler & Punch, 1997).

The algorithm has been applied to larger data sets in an experimental verification that the method is scalable. All the documents in the J series data sets (185 documents) were obtained by an Altavista search, whereas the documents in the K series data sets (2340 documents) were obtained through the Yahoo online news service. The J series data set includes pages in 10 broad categories, ranging from business capital to intellectual property, electronic commerce, information systems, affirmative action, employee rights, personnel management, industrial partnership, manufacturing systems integration, and materials processing.

The J1 and K1 data sets used all the words (minus stop words); the other sets were obtained by pruning the word list using different strategies, such as selecting only the most frequently occurring words from each document, or including also the words that have HTML tags. Details and results can be found in (Han et al., 1998a).

In Figure 2 we illustrate how cost behaves linearly with the number of nonzeros in the term frequency matrix. As each document uses only a small fraction of the entire dictionary of words, the term frequency matrix is very sparse. For example, in the K1 data set shown in Fig. 2, only 0.68% of the entries were nonzero. The PDDP algorithm is able to take advantage of this sparsity, yielding scalable performance.

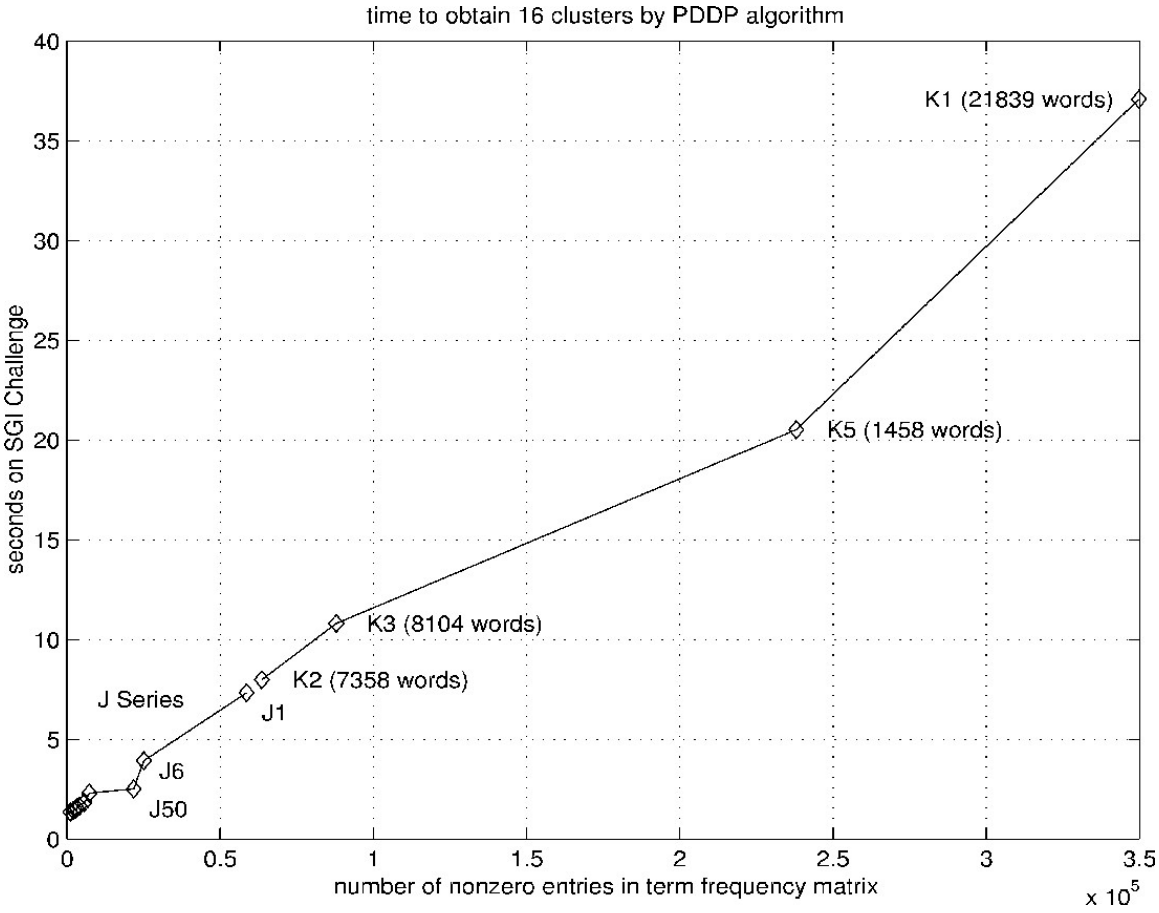

Figure 2: Time to obtain 16 clusters for various data sets using the PDDP Algorithm. The time in seconds on an SGI challenge (vertical axis) is plotted against the number of nonzeroes in the term frequency matrix (horizontal axis).

3.3 Distinctive Cluster Labels

The PDDP algorithm generates cluster labels consisting of distinctive words within each cluster, where the length of each label in words can be adjusted by the user. In Figure 3 we give a sample output of the cluster labeling module. This example was computed using the J1 set of 185 web documents, and each cluster was subdivided into three subclusters by means of two applications of the PDDP cluster splitting process. This branching factor of three as well as the number of levels to display is also adjustable by the user.

The distinctive words are not necessarily the words occurring the most often in the cluster, such as those represented in the cluster's centroid vector. Rather the distinctive words are obtained from the principal direction vectors computed by the PDDP method. These vectors yield the words that were most significant in choosing to place a document in one subcluster as opposed to another, each time a cluster is split (Boley, 1998).

```
94 technologi.system.develop
      46 system.manufactur.engin
            23 technologi.manufactur.integ
            12 inform.industri.research
            11 heate.process.materi
      20 industri.busi.loan
             9 industri.partnership.inform
             7 busi.loan.financ
             4 technologi.compani.fund
      28 electron.internet.web
            12 inform.system.manag
             8 electron.busi.phillip
             8 oracl.server.applic
68 patent.intellectu.properti
      25 patent.intellectu.properti
            12 patent.intellectu.properti
            10 copyright.copi.law
             3 web.internet.commerc
      29 busi.capit.manag
            11 busi.capit.financi
            12 personnel.manag.servic
             6 public.comment.certif
      14 union.employe.court
             5 union.disciplin.labor
             6 employe.dai.leave
             3 bargain.court.agreem
23 affirm.action.minor
      14 affirm.action.people
             3 raza.public.preferenti
             7 action.affirm.social
             4 black.white.women
       5 applic.minor.percent
             1 kolbe.instinct.action
             2 applic.school.student
             2 women.glass.minor
       4 employ.employe.innsmouth
             1 innsmouth.prejudic.look
             2 employ.employe.erisa
             1 colleg.southern.affirm
```

Figure 3: Hierarchical cluster labels generated by the PDDP algorithm. The number in front of each label is the number of documents in the cluster. understands is HTTP. Proxy servers are usually used to control access, but WebACE's proxy server simply forwards all requests after recording some data about the request. This data is used to generate the user profile.

4 Implementation of WebACE

WebACE is implemented as a browser independent Java application. Java was selected due to its full featured library for writing Internet applications and its ability to execute across a variety of platforms. The code was written using the JDK 1.1 and is entirely in Pure Java™ .

4.1 Design Issues

Several design issues were considered when constructing WebACE:

- simple to configure and use;
- non-intrusive - the user should not be forced to use the agent or interrupted while working. Ideally, users should not even be aware they are using an agent;
- responsive - using the agent should not significantly increase users' browsing time;
- cross-platform - the agent should be able to run on Unix, Windows NT, Windows 95, and Macintosh platforms. In addition, the agent should not force the user into a choice of browser. Unfortunately, WebACE's postprocessing and clustering modules are not written in Java, but mostly in Matlab. Because of this, the current version of WebACE is limited to running on machines using Sun Microsystems' Solaris operating system. Regardless, WebACE remains browser independent and can be used with Netscape, Microsoft Internet Explorer 4.0 for Unix, or any other browser running on Solaris and capable of being configured to use a proxy server. Nevertheless, it would be easy to port it to any platform supporting a standard web browser, Java and Matlab.

4.2 The Proxy Server

The heart of WebACE is a proxy server. A proxy server is an application layer program that acts as an intermediary between hosts on a network and the outside world. Proxy servers are usually used in conjunction with firewalls. A machine that is prevented from connecting to the external network by a firewall would request a service from the local proxy server, which would in turn request the service from the remote server and then forward the response back to the original requester. A proxy server has understanding of some number of application level protocols. In WebACE's case, the protocol it understands is HTTP. Proxy servers are usually used to control access, but WebACE's proxy server simply forwards all requests after recording some data about the request. This data is used to generate the user profile.

The proxy server leverages the ease of creating sockets in Java. Java provides a ServerSocket class to enable the writing of server applications. A server socket runs on a server and listens for incoming connections on a particular port. When a client socket attempts to connect to that port, the server negotiates the connection between the client and the server, and opens a regular socket between the two. Server sockets can handle multiple connections at a time. WebACE's server socket is created as follows:

```
// make a socket for the proxy server and set the port
ServerSocket servSock = new ServerSocket(8080);
```

WebACE's proxy server is a multi-threaded application. The thread architecture is shown in Figure 4. The main thread spawns the browser and starts a threads to listen for requests from the browser on a TCP/IP port. As TCP/IP sessions are accepted, this thread spawns request threads to handle the sessions. This allows multi-threaded browsers the opportunity to have more than one request pending at a time, speeding throughput. Request threads are short-lived, existing only for the duration of one HTTP request.

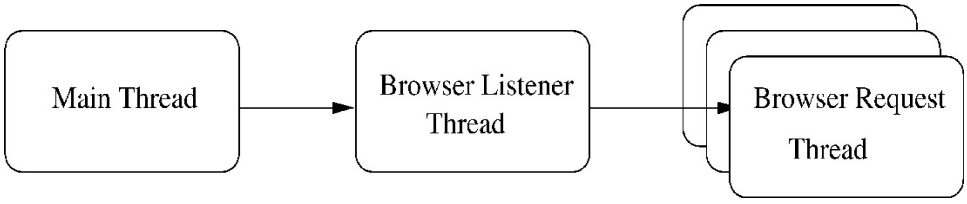

Figure 4: WebACE Threads

When the user enters a URL or clicks on a hyperlink, the browser sends an HTTP request to the proxy server. The proxy server parses the request into three parts as defined by HTTP/1.0 specification: request method, Uniform Resource Indicator (URI), and HTTP version. The requested document's URL is extracted from the URI, and then the request is reassembled and forwarded to the remote server. Once the remote server responds to the request, the response is parsed for the <title> HTML tag and then passed on to the browser for display. If the response is an HTML document, the agent records information about the document in the user's profile.

4.3 User Profile Creation

Two classes are used to record user profile information. The topmost class is the UserProfile class. At any given time there is only one object of the UserProfile class, which corresponds to the user's profile. The UserProfile class contains a vector of Document objects. At the next level is the Document object, which stores the URL, title, number of hits for that document, the start time of the most recent hit, the end time of the most recent hit, the total time spent at the document by the user, and that document's rank. The object model can be seen in Figure 5, which is drawn according to standard UML notation. This object graph is read from disk when the proxy server starts and written to disk when the proxy exits.

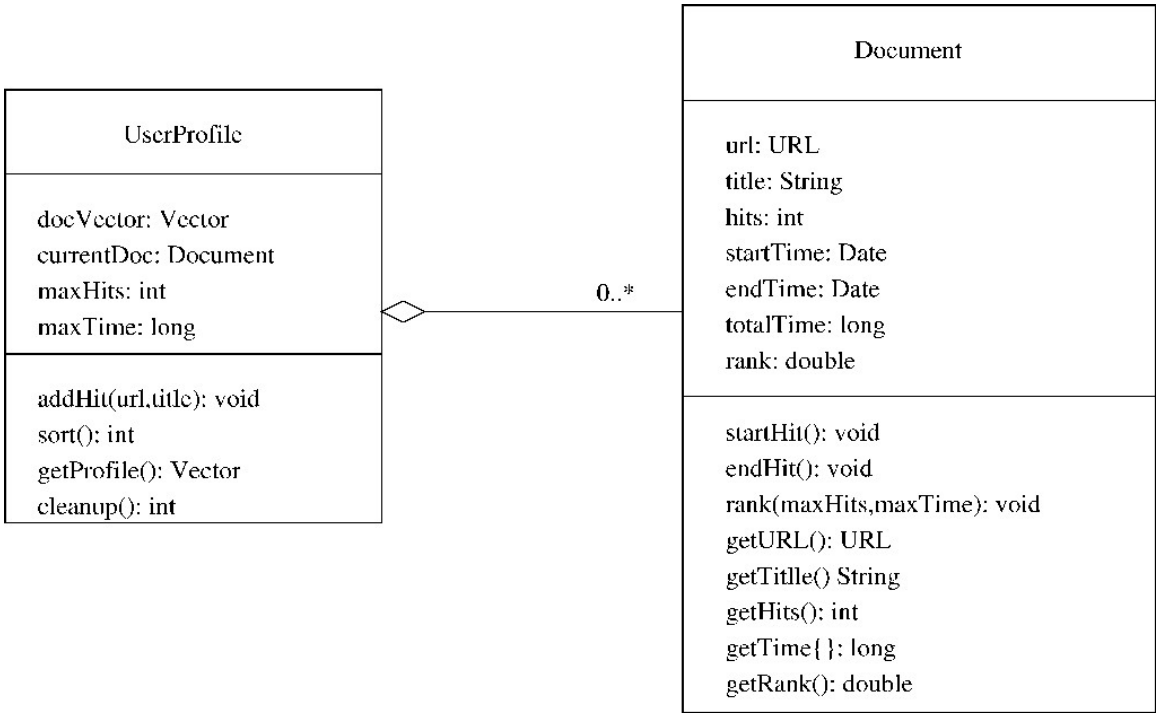

Figure 5: User Profile Object Model

When a user runs WebACE, the agent checks to see if a user profile exists. If not, a new profile is created and the first document the user visits is simply added to the profile. If a user profile does exist, when the user visits their first document WebACE checks the profile to see if the document, as keyed by its URL, is already in the profile. If the document is not in the profile, WebACE creates a Document object with the document's URL and title, sets the number of hits to one, sets start time to the current time, and inserts it at the end of the profile. If the document already exists in the profile, the agent simply increments that document's number of hits by one and sets the start time to the current time. When the user requests another document, the agent sets the end time for the old document to the current time and calculates the total time by subtracting the start time from the end time and adding it to the current total time, if any.

WebACE makes use of a reference to a Document object called currentDoc to keep track of the document the user is currently viewing. When WebACE adds a new document to the profile, or updates an existing document's fields, currentDoc is set to reference that document. When the user requests a new document from the proxy server, the agent calls the endHit() method on the document referenced by currentDoc. When the user exits the browser, WebACE traverses the user profile to find the document with the maximum number of hits and the document where the user spent the maximum amount of time. This information is recorded and used to determine each document's rank. A document's rank is calculated according to the following formula:

$$rank = \left(\left(\frac{hits}{\max\_hits}\right) \times 50\right) + \left(\left(\frac{time}{\max\_time}\right) \times 50\right)$$

The maximum time and hit information is used to normalize the ranks so that the maximum rank attainable is 100. Once ranks for all the documents have been calculated, the user profile is sorted by rank in descending order. While the user profile can theoretically handle an infinite number of documents, only the top one hundred ranked documents are passed to the clustering module for clustering and written to disk. If the total number of documents in the profile is less than one hundred, then all documents are clustered. This cut-off of one hundred documents is completely arbitrary and can easily be changed.

WebACE uses the JDK 1.1's Object Serialization feature to read in and write out the user profile. Object Serialization supports the encoding of objects, and the objects reachable from them, into a stream of bytes; and it supports the complementary reconstruction of the object graph from the stream. Serialization is used for lightweight persistence and for communication via sockets or Remote Method Invocation (RMI). The default encoding of objects protects private and transient data, and supports the evolution of the classes. The reading and writing of serializable objects is accomplished in the following manner:

```
// read the user profile from disk
ObjectInputStream ois = new ObjectInputStream(new
     FileInputStream("WebACE.db"));
userProfile = (UserProfile)ois.readObject();

// write the user profile to disk
ObjectOutputStream oos = new ObjectOutputStream(new
      FileOutputStream("WebACE.db"));
oos.writeObject(userProfile);
oos.close();
```

4.4 Post-processing

Once the user profile has been sorted, WebACE traverses the profile again and streams the documents' URLs to a post-processing script that reduces the documents to vectors and produces a term frequency matrix in sparse form. The post-processing script was written by Jerry Moore and is implemented in Perl. This script parses each document into its composite words. These words are run through a stop-list where words with no useful semantic meaning are removed. The remaining words are then "stemmed" or have their endings removed, leaving only the base of the word. Stemming is done using Porter's suffix-stripping algorithm (Porter, 1980) as implemented by (Frakes, 1992). This process is repeated for every document in the user profile. When all the documents have been processed, the script writes a file containing a sparse form matrix to disk. Because the post-processing script is a separate program and is not written in Java, WebACE uses Java's Process class to execute the script. The agent then opens an OutputStream to the script and writes the URLs in the user profile to the stream. Upon completion of this task, WebACE waits for the post-processing script to finish execution before invoking the clustering module. The code for this process is listed below:

```
// this section of code runs the post-processing script that
// generates the matrix
Process jerry = Runtime.getRuntime().exec("agent");
OutputStream os = jerry.getOutputStream();
PrintWriter out = new PrintWriter(os);
Enumeration e =
    ((myWebACE.userProfile).getProfile()).elements();
while(e.hasMoreElements()) {
    Document d = (Document)e.nextElement();
    // stream the URLs to the post-processing script
    out.println((d.getURL()).toString());
}
out.close();
   // wait for the post-processing script to finish before
   // running the clustering module
jerry.waitFor();
```

4.5 Obtaining and Presenting the Clustering

The prototype WebACE agent has been implemented as a proxy server residing on the same host as the user. The user simply browses the Web using a browser configured to use the given agent as a proxy for http requests. The prototype agent simply records the documents fetched by the user with the browser in the natural way. After the user quits the browser, the agent begins the post-processing script: the documents are clustered using an unsupervised clustering method, and the results of the clustering are collected into a web page of URLs grouped into clusters, shown in Figure 6.

The clustering process consists three steps: (a) the documents are sorted and pruned to the 100 most interesting documents based on some heuristics (sec. 4.3), (b) the word counts for each document are collected (sec. 4.4), and (c) a clustering algorithm is applied (sec. 3).

There are currently two clustering modules available for use by WebACE. The first uses principal direction divisive partitioning (PDDP) written in Matlab, as described in Section 3.1. The second

implements association rule hypergraph partitioning (ARHP) and was written in a combination of C, Perl, Matlab, and awk (Han et al., 1998b). ARHP first finds sets of documents that have many words in common using association rule discovery methods (Agrawal et al., 1996). These frequent item sets are then used to group documents into hypergraph edges, and a hypergraph partitioning algorithm (Karypis et al., 1997) is used to find document clusters.

Both clustering modules read in the matrix file produced by the post-processing script and output a file containing document clusters, without any user input.

The WebACE graphical display consists of two frames as shown in Figure 6. In the left frame, the agent displays the clusters. This frame is automatically generated by WebACE following the completed execution of the clustering module. The clusters are presented as a list of document titles, and are separated by horizontal rules. In the case where a document does not have a title, its URL will be listed instead. The right frame is the display window. The user can click on any of the documents in the cluster display frame, and its content will be shown in the display window.

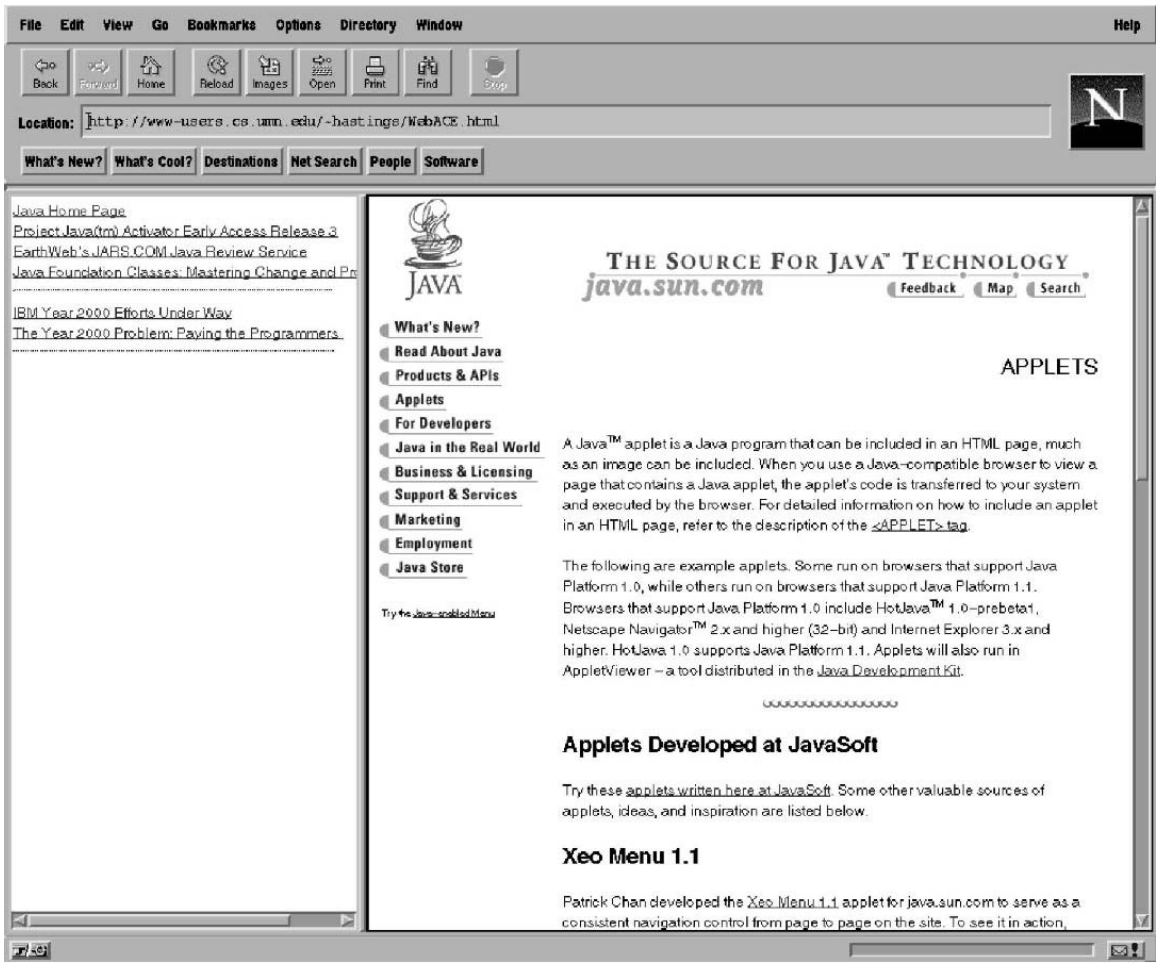

Figure 6: WebACE Graphical Display

5 Related Work

A number of Web agents use various information retrieval techniques (Frakes & Baeza-Yates, 1992) and characteristics of hypertext Web documents to automatically retrieve, filter, and categorize these documents (Chang & Hsu, 1997, Broder, Glassman, & Manasse, 1997, Maarek & Shaul, 1996, Wulfekuhler & Punch, 1997, Weiss et al., 1996). For example, HyPursuit (Weiss et al., 1996) uses semantic information embedded in link structures as well as document content to create cluster hierarchies of hypertext documents and structure an information space. BO (Bookmark Organizer) (Maarek & Shaul, 1996) combines hierarchical agglomerative clustering techniques and user interaction to organize collection of Web documents listed in a personal bookmark file.

Pattern recognition methods and word clustering using the Hartigan's K-means partitional clustering algorithm are used in (Wulfekuhler & Punch, 1997) to discover salient HTML document features (words) that can be used in finding similar HTML documents on the Web. Broder (Broder, Glassman, & Manasse, 1997) calculates a sketch for every document on the web and then clusters together similar documents whose sketches exceed a threshold of resemblance. Given a document's URL, similar documents can be easily identified, but an index for the whole WWW needs to be maintained.

Maarek (Maarek & Shaul, 1996) uses the Hierarchical Agglomerative Clustering method to form clusters of the documents listed in a personal bookmark file. Individual documents are characterized by profile vectors consisting of pairs of lexically affine words, with document similarity a function of how many indices they share. This method may not scale well to large document searches.

The Syskill & Webert system (Pazzani, Muramatsu, & Billsus, 1996) represents an HTML page with a Boolean feature vector, and then uses naive Bayesian classification to find web pages that are similar, but for only a given single user profile. Balabanovic (Balabanovic, Shoham, & Yun, 1995) presents a system that uses a single well-defined profile to find similar web documents for a user. Candidate Web pages are located using best-first search, comparing their word vectors against a user profile vector, and returning the highest -scoring pages. A TFIDF scheme is used to calculate the word weights, normalized for document length. The system needs to keep a large dictionary and is limited to one user.

The Kohonen Self-Organizing Feature Map (Kohonen, 1988) is a neural network based scheme that projects high dimensional input data into a feature map of a smaller dimension such that the proximity relationships among input data are preserved. On data sets of very large dimensionality such as those discussed here, convergence could be slow, depending upon the initialization.

6 Conclusion and Future Work

We have proposed an agent to explore the Web, categorizing the documents a user reads and then using automatically generated categories to assist the user in further exploring the Web. We have presented the architecture of the agent, described its design and implementation, and given some examples showing the cluster labels generated by the clustering algorithm.

By designing a client-side agent we avoid the bottleneck that would arise from accessing remote servers and we give the user complete privacy and local control of hiw own profile. This also

allows an easy interface with the user's own bookmarks and, in principle. the clustering algorithm could be used to organize the bookmarks. In addition, the clustering algorithm ability to generate taxonomies could also be used to annotate bookmarks.

The unsupervised nature of the clustering algorithms releases the user of the burden of supplying keyword or categories in advance. This is particularly useful given the massive amount of information available on the Web. The user can accumulate documents over an extended period of time, without having to remember the categories or topics from those documents.

In extensive experiments, both clustering algorithms, PDDP and ARHP, have been found to yield high quality clusters, as compared to AutoClass and hierarchical agglomerative clustering (Han et al., 1998a).

Currently only the main components of the agent have been implemented by using a proxy server, enough to create a single-user client-side server prototype capable of clustering documents retrieved by the user.

In the future, we will explore the performance of the agent as an integrated and fully automated system, comparing the relative merits of various algorithms for clustering, query generation, and document filtering, when used as the key components for this agent.

One of the components we will add to the agent is the ability to search the Web for documents that are related to the clusters of documents already found by the user. The key question here is how to select from the existing clusters a representative set of words to be used in the Web search. We have done some initial experiments using a combination of the words that occur frequently across documents and in individual documents and of the words that occur frequently in individual documents but not across documents. Preliminary results, reported in (Han et al., 1998a), show that this method is capable of producing small sets of relevant documents using standard search engines.

Our future research plans include also developing new methods for incremental clustering of documents after discovering an initial set of clusters, or recognizing when the user's interests change. The idea is the following: the binary tree can be used to filter new incoming documents by placing each new document on one or the other side of the root hyperplane, then placing it on one or the other side the next appropriate hyperplane, letting percolate down the tree until it reaches a leaf node. We then know that the new incoming document is most closely related to the documents already in that cluster. If the combined scatter value with the new document is too large, then we know the new document is only loosely related to the cluster and we need to split it again.

Acknowledgements

This research was partially supported by NSF grants CCR 9405380 and 9628786.

References

Ackerman, M.; Billsus, D.; Goffney, S.; Hettich, S.; Khoo, G.; Kim, D. J.; Klefstad, R.; Lowe, C.; Ludeman, A.; Mutamatsu, J.; Omori, K.; Pazzani, M.; Semler, D.; Starr, B.; and Yap, P. (1997), "Learning probabilistic user profiles". AI Magazine, Vol 18, No. 2,:pp 47-56.

Agrawal, A.; Mannila, H.; Srikant, R.; Toivonen, H.; and Verkamo, A. (1996), "Fast discovery of association rules", In Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R., eds., *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press. pp 307-328.

Armstrong, R.; Freitag, D.; Joachims, T.; and Mitchell, T. (1995), "WebWatcher: A learning apprentice for the World Wide Web", In Proc. AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments. AAAI Press.

Balabanovic, M.; Shoham, Y.; and Yun, Y. (1995), "An adaptive agent for automated Web browsing". Journal of Visual Communication and Image Representation, Vol. 6, No. 4.

Berry, M. W.; Dumais, S. T.; and O'Brien, G. W. (1995), "Using linear algebra for intelligent information retrieval". SIAM Review, Vol. 37, pp 573-595.

Boley, D. (1997),"Principal Direction Divisive Partitioning". Technical Report TR-97-056, Department of Computer Science, University of Minnesota, Minneapolis, to appear in Data Mining and Knowledge Discovery.

Boley, D. (1998), "Hierarchical taxonomies using divisive partitioning". Technical Report TR-98-012, Department of Computer Science, University of Minnesota, Minneapolis.

Broder, A. Z.; Glassman, S. C.; Manasse, M. S.; and Zweig, G. (1997), "Syntactic clustering of the Web". In Proc. of 6th International World Wide Web Conference.

Chang, C., and Hsu, C. (1997), "Customizable multi-engine search tool with clustering". In Proc. of 6th International World Wide Web Conference.

Cheeseman, P., and Stutz, J. (1996), "Bayesian classification (Autoclass): Theory and results". In Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R., eds., *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press. pp 153-180.

Dubes, R., and Jain, A. (1980), "Clustering methodologies in exploratory data analysis". In Yovits, M., ed., Advances in Computers. New York: Academic Press Inc.

Duda, R. O., and Hart, P. E. (1973), *Pattern Classification and scene analysis*. John Wiley & Sons.

Fisher, D. (1995), "Optimization and simplification of hierarchical clusterings." In Proc. of the First Int'l Conference on Knowledge Discovery and Data Mining, pp 118-123.

Frakes, W. B., and Baeza-Yates, R. (1992), *Information Retrieval Data Structures and Algorithms. Englewood Cliffs*, NJ: Prentice Hall.

Frakes, W. B. (1992), "Stemming algorithms". In Frakes, W. B., and Baeza-Yates, R., eds., *Information Retrieval Data Structures and Algorithms*. Prentice Hall. pp. 131-160.

Golub, G. H., and Van Loan, C. F. (1996), *Matrix Computations*. Johns Hopkins Univ. Press, 3rd edition.

Han, E. H. S.; Boley, D.; Gini, M.; Gross, R.; Hastings., K.; Karypis, G.; Kumar, V.; Mobasher, B.; and Moore, J. (1998a), "WebACE: A Web agent for document categorization and exploration". In Proc. of 2nd International Conference on Autonomous Agents.

Han, E.; Karypis, G.; Kumar, V.; and Mobasher, B. (1998b), "Hypergraph based clustering in high-dimensional data sets: A summary of results". Bulletin of the Technical Committee on Data Engineering, Vol. 21, No. 1.

Jain, A., and Dubes, R. C. (1988), *Algorithms for Clustering Data*. Prentice Hall.

Karypis, G.; Aggarwal, R.; Kumar, V.; and Shekhar, S. (1997), "Multilevel hypergraph partitioning: Application in VLSI domain". In Proceedings ACM/IEEE Design Automation Conference.

Kohonen, T. (1988), *Self-Organization and Associated Memory*. Springer-Verlag.

Lee, R. (1981), Clustering analysis and its applications. In Toum, J., ed., *Advances in Information Systems Science*. New York: Plenum Press.

Lu, S., and Fu, K. (1978), "A sentence-to-sentence clustering procedure for pattern analysis". IEEE Transactions on Systems, Man and Cybernetics, Vol. 8, pp 381-389.

Maarek, Y. S., and Shaul, I. Z. B. (1996), "Automatically organizing bookmarks per contents". In Proc. of 5th International World Wide Web Conference.

Ng, R., and Han, J. (1994), "Efficient and effective clustering method for spatial data mining". In Proc. of the 20th VLDB Conference, pp 144-155.

Pazzani, M.; Muramatsu, J.; and Billsus, D. (1996), "Syskill & Webert: Identifying interesting Web sites". In National Conference on Artificial Intelligence, Portland OR, pp 54-61.

Porter, M. F. (1980), "An algorithm for suffix stripping". Program, Vol. 14, No. 3, pp 130-137.

Salton, G., and McGill, M. J. (1983), *Introduction to Modern Information Retrieval*. McGraw-Hill.

Shavlik, J., and Dietterich, T. (1990), *Readings in Machine Learning*. Morgan-Kaufman.

Titterington, D.; Smith, A.; and Makov, U. (1985), *Statistical Analysis of Finite Mixture Distributions*. John Wiley & Sons.

Weiss, R.; Velez, B.; Sheldon, M. A.; Nemprempre, C.; Szilagyi, P.; Duda, A.; and Gifford, D. K. (1996), "Hypursuit: A hierarchical network search engine that exploits content-link hypertext clustering". In Seventh ACM Conference on Hypertext.

Wulfekuhler, M. R., and Punch, W. F. (1997), "Finding salient features for personal Web page categories". In Proc. of 6th International World Wide Web Conference.