Mining Navigation History for Recommendation

Xiaobin Fu, Jay Budzik, Kristian J. Hammond
Infolab, Northwestern University
1890 Maple Avenue
Evanston, IL 60201
1(847) 467-1265
fu, budzik, hammond@ils.nwu.edu

ABSTRACT

Although a user's navigation history contains a lot of hidden information about the relationship between web pages and between users, this information is usually not exploited. The information hidden in the history can be an invaluable source of knowledge in assisting a user to better surf the Web. We presented a system which actively monitors and tracks a user's navigation. Once a user's navigation history is captured, we apply data mining techniques to discover the hidden knowledge contained in the history. The knowledge is then used to suggest potentially interesting web pages to users.

Keywords

Data mining, Collaborative Information Recommendation, Intelligent User Interface.

1. MOTIVATION

Although a user's navigation history contains a lot of hidden information about the relationship between web pages and between users, this information is usually not exploited. Some collaborative information recommending systems require users to rate a web page, store these ratings and make recommendations based on the ratings[7, 9]. While this method is a good approach in capturing a user's reading experience, it requires extra work on the part of the user. The performance of such systems depends on how well users use them.

Another method of reusing a user's reading experience is to ask a user to input related web pages to the page being read. The user's recommended pages will then be used by other users. For example, a system called "Alexa"[3], employs such a method.

These methods are onerous to many users. In fact, the footprints a user leaves while surfing the Web contain enough information for us to discover hidden knowledge about interesting pages, even without user's explicit ratings or inputs. Our research goal reported in this paper is to actively but quietly keep tracks of what a user has read, to collect that user's navigation history in a central repository, and apply data mining techniques to discover hidden information from user's navigation history. There are two types of hidden knowledge discovered:

1. What sets of URLs are read together by many users? Statistically, there must be some relationship between such a set of URLs, whether they are similar to each other, or if they are separate components of a major topic.
2. What sets of users read many of the same web pages? If two users have read a lot similar page, we can conclude these two users have similar interests.

These kinds of knowledge are then used to make recommendation. The knowledge is gained directly from the user's navigation history, without any extra work on the part of the users. As we will discuss in later sections, this offers another method of collaborative information filtering.

We implement a system called "SurfLen" to use this method. SurfLen is an information recommendation system which suggest interesting web pages to users. The underlying data mining technique of SurfLen is "Association Rules".

2. ASSOCIATION RULES

One of the more well studied problems in data mining is the mining of association rules [1, 2, 4, 5] in market basket data, which was first introduced by Argawal, Imielinski and Swami [1, 2]. Market basket data stores items purchased on a per-transaction basis for a sales organization. Basket data type transactions do not necessarily consist of items bought at the same time. The data may be aggregated from items bought by a customer over an interval. The goal is to discover buying patterns in the form of rules like "90% of transactions that included purchasing bread and butter also included purchasing milk". Such rules can then be used for decision support, e.g., price promotion and store layout. This technique has proven highly successful for extracting useful information from large databases. Especially in recent years, a number of efficient algorithms such as "a-priori"[1, 2] have been presented for fast discovery of associations in large databases.

To place our work in the context of earlier work, we introduce some basic concepts of mining association rules to market basket data, using the formalism presented by Agrawal et al. Let $I = \{i_1, i_2, \ldots i_k\}$ be a set of binary attributes, called items. Each transaction T is represented as a set of items such that $T \subseteq I$. Let D be a set of such transactions. Let X be a set of s items such that $X \subseteq I$, called an s-itemset. We say that a transaction T contains X, if $X \subseteq T$. An association rule is an expression of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$.

We define *Support*($X \Rightarrow Y$) as the percentage *s%* if *s%* of transactions in *D* contain $X \cup Y$, and *Confidence*($X \Rightarrow Y$) as the percentage *c%* if *c%* of transactions in D that contain *X* also contain *Y*. An association rule $X \Rightarrow Y$ holds for D if *Support*($X \Rightarrow Y$) > *minsup* and *Confidence*($X \Rightarrow Y$) > *minconf*, where *minsup* and *minconf* are two threshold values set in the system. Support and Confidence are two important measures of association. We can also represent these two measures as *Support*($X \cup Y$) and *Confidence*($X \cup Y$). Usually we are only interested in those association rules that have high support or confidence.

An association rule captures item dependency in a transaction set *D*. Essentially, it answers a question: what items do customers often buy together? Interestingly enough, one grocery store observed that people who buy diapers also buy beer. The association rule *diapers* ⇒ *beer* has high support and confidence because large fractions of all customers who buy diapers also buy beer. Once association rules like this are found, they can be used in a variety of ways. For example, knowledge of the rule *diapers* ⇒ *beer* could enable the store to put diapers on sale, and keep beer at regular price and consequently predict the volume of sales for both diapers and beer would be increased during the sale period.

To find all the association rules $X \Rightarrow Y$ with *Support*($X \Rightarrow Y$) > *minsup*, the algorithm we use performs two steps. First, all itemsets with *minsup* (called *large* itemsets) are generated from the database. This is a computationally expensive step. In the second step, association rules are generated from these large itemsets. This step is very straightforward. For a given *large* itemset $X = \{ i_1, i_2, \ldots i_k \}$, $K \geq 2$, the antecedent of each association rule will be a subset *Y* of *X*, and the consequent will be the subset $X - Y$. These rules can be further constrained. For example, one can specify that the number of items in a rule consequent is 1.

Most current algorithms for the discovery of large itemsets work iteratively. They first query the database to find all the large 1-itemsets. Then the large *s*-itemsets are generated from the large (*s-1*)-itemsets of the previous pass. Since the process of discovering large itemsets is expensive over large databases, various techniques have been proposed to speed up the search for large itemsets. One of the important optimizations is called "a-priori".

2.1 A-Priori Optimization

When generating candidate itemsets, one of the most important observations is that any subset of a large itemset must be large [2]. Therefore, the candidate itemsets having *k* items can be generated by joining large itemsets having *k*-1 items. This can result in much smaller number of candidate itemsets. For example, if we are looking for pairs of items with *minsup*, we can only consider those items that appear in the database having *minsup*. Provided *minsup* is high enough, the number of items for the next joining step will be small enough to speed up the computation significantly.

3. COLLABORATIVE RECOMMENDATION SYSTEM

The World Wide Web provides a vast resource of information and services that continues to grow rapidly. The heterogeneity and the lack of structure have made it very difficult for users to discover new interesting information. In an effort to alleviate people's information overload, many searching and indexing tools such as Yahoo and Lycos [10, 11] have emerged. Unfortunately, searching has been problematic due to many users' difficulty in constructing search queries. Because of this difficulty, the number of results returned by a search engine is often huge, making it nearly impossible to locate appropriate information. Furthermore, discovery is different from searching in that searching is goal-directed, i.e., it is an activity users engage in when they have a specific information need, while discovery is often accidental.

Recently systems that recommend information to users have gained much attention. A recommender system suggests new information to a user based on that user's preferences and past behavior. Such systems can help users discover new information relating to their areas of interest. One popular technique is to build collaborative information recommender systems [7, 9]. The basic idea behind these systems is that people will tend to agree with the evaluation of a certain information item if they have displayed a pattern of past agreements over other information items. It makes use of the opinions of people who have already seen a piece of information for people who have not yet seen it. In fact, we use collaborative recommendation everyday in an informal way: we often forward an article to a friend; we suggest a new type of car to a colleague, etc.

A system employing this technique usually requires users to explicitly input ratings about a piece of information. These ratings are then used to compute pairwise correlation coefficients among existing users. The correlation coefficient is the measure of how similar two users are. The system can make predictions or recommendations based on the correlation coefficients: if User A and User B have a high correlation coefficient, then it is highly possible that User A's rating of an item can be used to predict User B's interest in it as well.

The current popular algorithm to compute correlation coefficients is *Pearson r* Correlation Coefficient. Given two users' list of ratings as $X = [x_1, \ . \ . \ . \ , x_t]^T$ and $Y = [y_1, \ . \ . \ . \ y_t]^T$, the standard *Pearson r* correlation coefficient is used to measure the similarity ($\boldsymbol{S_r}$) between two lists of ratings. It is calculated as:

$$\frac{\sum_{i=1}^{t}\left(x_i - x_{avg}\right)\left(y_i - y_{avg}\right)}{\sqrt{\left(\sum_{i=1}^{t}\left(x_i - x_{avg}\right)^2\right)\left(\sum_{i=1}^{t}\left(y_i - y_{avg}\right)^2\right)}}$$

Our research extends association rule mining to collaborative recommender systems. Instead of evaluating similarity between users, we compute dependencies among all the information items that a user has seen. There are several advantages of this method over the above method. First, it can handle a user's multiple and varied interests. The *Pearson r* Correlation Coefficient works best when the information space contains stable homogeneous information contents. Second, it does not require users to input ratings explicitly. This is a great benefit because rating requires extra effort on the part of the users.

4. SURFLEN: OVERVIEW AND IMPLEMENTATION

Our system SurfLen works as follows. It watches over a user's shoulder. Each URL browsed by the user is stored as a part of the user's browsing history. As soon as a user accesses a new Web page, SurfLen connects to a central recommender engine, and gets a list of related web pages. A popup window displays links to these pages.

Given the browsing history database *D*, each row of records is a <*u*, *p*> pair, where *u* is a numeric identifier of the corresponding user, and *p* is a numeric identifier of the corresponding URL. Each user *U* can be represented as a set of URLs $\{p_1, p_2, \ldots, p_m\}$, and each URL can be represented as a set of users $\{u_1, u_2, \ldots, u_m\}$. Using the proven algorithm "a-priori", we can find association rules like $\{p_1, p_2, \ldots, p_m\} \Rightarrow p$, *p* will then be recommended to users who have read $\{p_1, p_2, \ldots, p_m\}$ but not *p*.

Suppose a set of 2-item URL sets $\{\{p, p_1\}, \{p, p_2\}, \ldots, \{p, p_n\}\}$, and a user is reading *p*, then, unread items in $\{p_1, p_2, \ldots, p_n\}$ will be recommended to this user. Again, the algorithm runs iteratively through each 2-item URL set and gets the list of recommendations.

Straightforward application of the above method poses several problems in a space as large as the web. One problem is dealing with sparse datasets: if users browsing histories intersect infrequently, the system on its own cannot make recommendations. To alleviate this problem, we introduce an algorithm for ranking the association rules generated by the system. This allows it to provide recommendations even when the degree of intersection between a user's history and a given set of rules is small. The idea is to rank each itemset based on the number of intersecting URLs between the itemset and the user's browsing history, and use the best *k* itemsets for recommendation. Given itemsets *S*, and a collection of user browsing histories *U*, for each history $U_i = \{p_1, p_2, p_3, \ldots, p_m\} \in U$, and itemset $S_j = \{p_1', p_2', p_3', \ldots, p'_m\} \in S$, we define $rank(S_j, U_i) = |S_j \cap U_i|$. Itemsets are then stored in a heap sorted by this rank, and the top *k* can be used for making recommendations.

Another solution to this problem is to find large 2-item user sets $\{u_1, u_2\}$. Given this, URLs that have been read by user u_1 can be recommended to user u_2, and vice versa. The user u_1 and user u_2 are called *close neighbors*. In addition, work on extending the above algorithm to include continuous-valued document comparison functions based on content is in progress.

The heart of SurfLen is the SurfLen Recommendation Engine (SRE), which continuously searches for association rules. Each SurfLen client installed at an end user's machine connects to SRE and updates the user's browsing history. In return, SRE recommends potentially interesting URLs. The SurfLen client displays the list of URLs in a popup dialog box. The architecture is outlined in figure 1.

4.1 Surflen Client

The SurfLen client interfaces between SRE and Microsoft Internet Explorer. It is implemented as a Microsoft Browser Helper Object (BHO) [12], a COM component that Internet Explorer loads each time it starts up. A BHO can intercept the browser's typical events, such as the "Download Complete" event and monitor a user's action. As a BHO, SurfLen does not change to the user's browser interface significantly. Due to architectural limitations, we have not implemented the same type of client for other browsers, i.e. Netscape Navigator and NCSA Mosaic, though access to the source code of these browsers does make this a possibility. Figure 2 shows a snapshot of the system integrated into Internet Explorer as a user is reading a web page on Distance Learning. SurfLen is recommending related web sites.

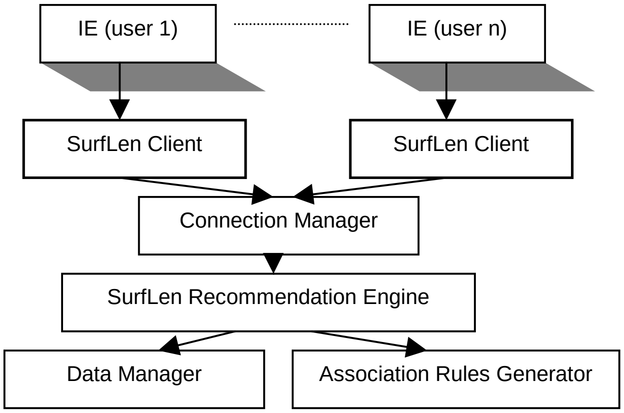


Figure 1: System Architecture

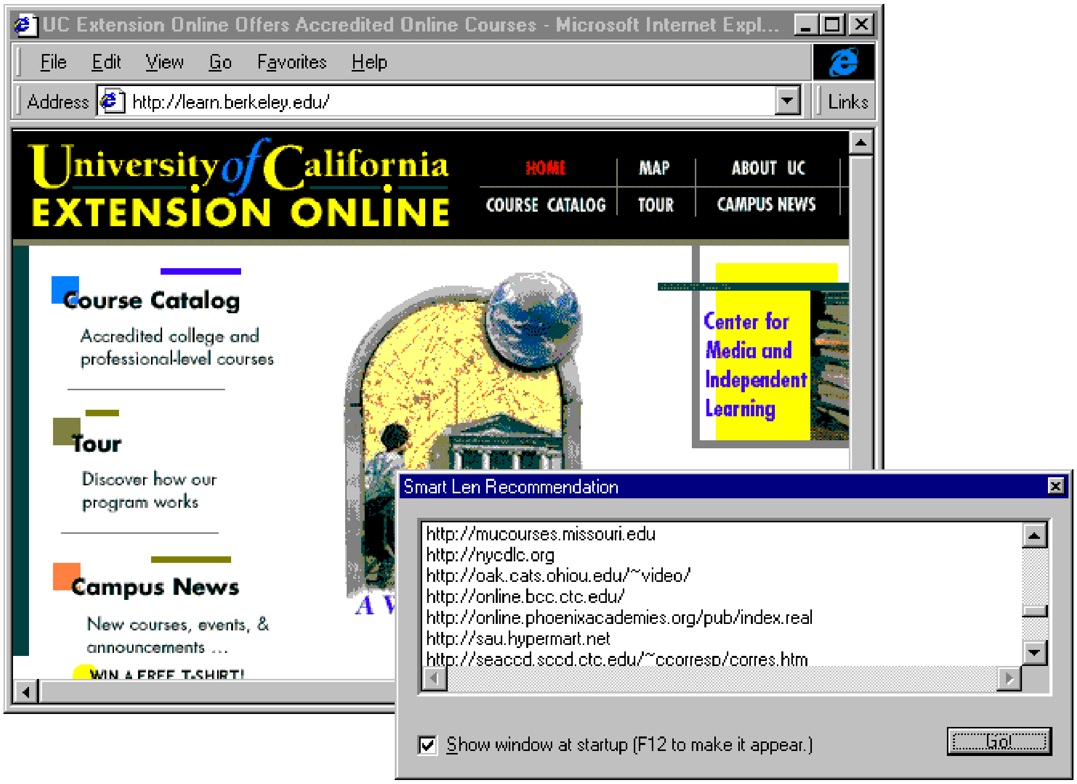
A simple protocol is established for the SurfLen Client to communicate with SRE. The protocol allows the client to communicate history information as well as retrieve recommendations from the system. There are three primary communication primitives:

- ini: Open an account with the SRE. This communication primitive is normally used when a user installs a SurfLen Client. The SRE will generate a unique user ID for the each new user which will be stored in the registry on the client machine.
- put: Send a URL to the SRE and get URLs related to the current page. Every time a user accesses a URL, the SurfLen client sends "put" command to the SRE.
- get: Get a list of recommended URLs. This command is used when the SurfLen Client is loaded by Internet Explorer.

The following algorithm describes the SurfLen Client's workflow when loaded by Internet Explorer. It ensures the recommendations made by the system are synchronized with the user's interactions in their browser:

1. Detect if SurfLen is being used for the first time by a user. If yes, read the user's bookmarks, parse into separate URLs and send these URLs to the SRE as the user's initial browsing history, using the communication primitive "ini".
2. After the SurfLen client is loaded by Internet Explorer, query the SRE to get a list of recommended URLs, and display them in a popup window, using the communication primitive "get".
3. Intercept the "Download Complete" event for each URL that a user accessed, send this URL to the SRE to update the user's browsing history, in return, get a list of recommended URLs related to this URL, using the communication primitive "put".

The following components of the system work in concert to manage incoming and outgoing information in the form of a user's browsing history, as well as new recommendations.


UC Extension Online Offers Accredited Online Courses - Microsoft Internet Expl...
File
Edit
View
Go
Favorites
Help
Address
http://learn.berkeley.edu/
Links
University of California
EXTENSION ONLINE
HOME
MAP
ABOUT UC
COURSE CATALOG
TOUR
CAMPUS NEWS
Course Catalog
Accredited college and professional-level courses
Tour
Discover how our program works
Campus News
New courses, events, & announcements ...
WIN A FREE T-SHIRT!
Center for Media and Independent Learning
Smart Len Recommendation
Go!

http://mucourses.missouri.edu
http://nycdlc.org
http://oak.cats.ohiou.edu/~video/
http://online.bcc.ctc.edu/
http://online.phoenixacademies.org/pub/index.real
http://sau.hypermart.net
http://seaccd.sccd.ctc.edu/~ccorresp/corres.htm

☑ Show window at startup (F12 to make it appear.)

Figure 2: SmartLens is recommending URLs as a user is browsing.

4.2 Other Components

Connection Manager: The Connection Manager handles incoming requests from the SurfLen Clients, and forks a new thread for each request The SurfLen client interfaces between SRE and Microsoft.

Data Manager: The Data Manager is responsible for storing URLs accessed by users. The underlying database management system is Microsoft SQL Server 7.0. The Database Manager connects the SQL server through ODBC.

Association Rules Generator: The Association Rules Generator implements the "a-priori" algorithm. SRE calls the generator on two different occasions. One is at midnight to generate *n*-itemsets, since generating *n*-itemsets is often time consuming (see the "association rules" section for a discussion of how itemsets are generated). The other is to generate 2-itemsets when a user's client requests related URLs for a particular URL being read. Given a minimum support threshold that is high enough, this step can be real-time.

SurfLen Recommendation Engine(**SRE**): SRE works like a broker. It coordinates activities of all the components of SurfLen. The following steps describe how SRE works.

1. Invoke the association rule generator to generate all the large URL itemsets and user itemsets at midnight, and store these itemsets.

2. Switch (incoming command):

   2.1. case "ini": Store the history URLs. Search the stored URL itemsets to find possible recommended URLs. Generate a user ID. Store the User ID into user table. Send the user ID and all the recommended URLs back to the SurfLen Client.

   2.2. case "put": Store the URL. Search the stored URL itemsets to find URLs related to this URL. If no URL found, call the association rule generator with a lower minimum support threshold. Send all the recommended URLs back to the SurfLen Client.

   2.3. case "get": Match the list of browsed URLs against the stored URL itemsets to find all the recommended URLs. If the number of the list is too small, get some possible URLs from the user's close neighbors (see the beginning of this section for the definition of close neighbor). Send these URLs back to the SurfLen client.

5. EVALUATION

The empirical evaluation of any large-scale collaborative recommender system is difficult because the system's performance is so highly dependent on broad usage patterns over

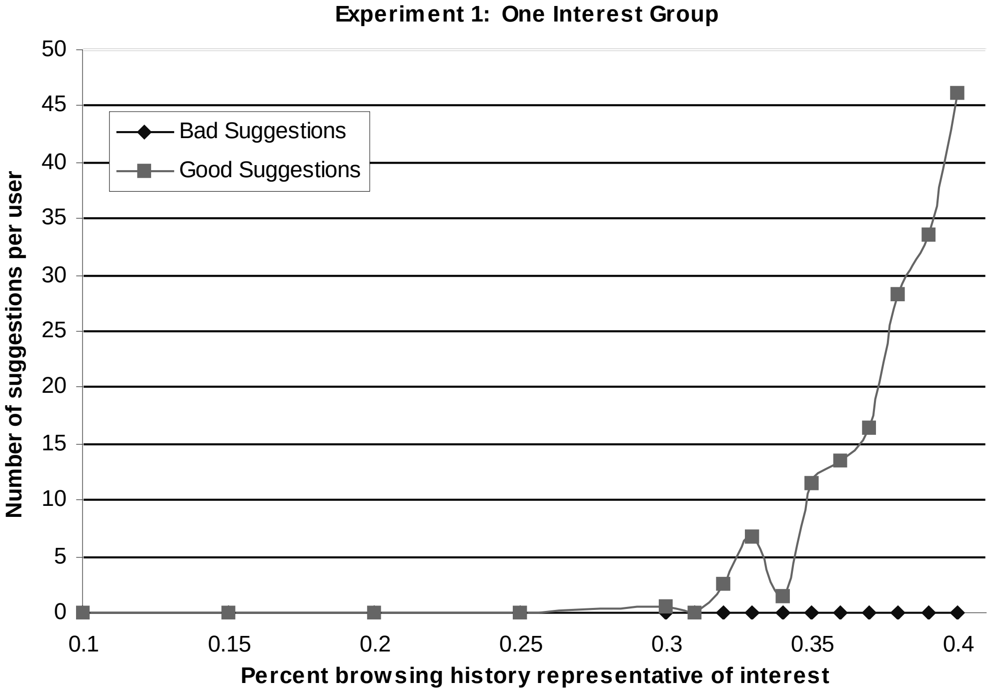


Figure 3: Results of Experiment 1. The number of good suggestions per user increases as their browsing history becomes more representative of their interests.

large groups of users. In light of this, and as a preliminary step in a full-scale empirical evaluation, we chose the following evaluation methodology. To evaluate the recommendations made by the system, we gathered three non-intersecting sets of URLs from Yahoo! [11]: one in Distance Learning, one in Finance, and the other a random set. We then used these URLs to construct groups of simulated users with specific interests (represented by the URLs in their history) by randomly choosing a percentage of URLs from each semantic category. For example, one test set attempted to simulate users interested in Distance Learning. These users' browsing histories were constructed by choosing 25% of the URLs from the Distance Learning pool, and the other 75% from the random pool.

We ran two experiments. The first was to determine the effect of noise on recommendations. Given that users have ephemeral interests, as well as the fact that viewing a page may not always be predictive of interest in the subject of that page (i.e., when a user clicks on the wrong link, or is mislead by a link label). The aim of this experiment therefore was to show that the accuracy of recommendations increases as the amount of noise decreases. This experiment shows first that if there is sufficient intersection across user browsing history, the system will produce valid suggestions. Furthermore, given the assumption that over time a user's browsing history tends to converge on an accurate representation of their interests, this experiment shows that the more a user interacts with the system, the better its recommendations will be.

For this experiment, we built browsing histories for an aggregate of 100 simulated users by drawing a fixed percentage from the list of URLs on Distance Learning, and the rest from a random pool of fixed size. The set of URLs on Distance Learning contained 100 URLs, while the random pool contained 400. At each iteration, the system made recommendations for each user. A recommendation was deemed relevant if it was new to the user and was from the set of Distance Learning URLs. The results of this experiment are shown in Figure 3. The graph shows that as the user's browsing history becomes more representative of their interests, the number of good recommendations per user increases.

The second experiment was aimed at studying the effect of common noise among interest groups. In this experiment, we constructed two aggregates of users: one interested in Distance Learning, the other in Finance. For each user, a fixed percentage of URLs was drawn randomly from the list of URLs from their interest group. The rest was drawn at random from a fixed pool of other URLs. This "other" group was kept constant for both groups of users, regardless of their interest. Each pool of interest (Distance Learning or Finance) contained 100 URLs, while the "other" pool contained 400 URLs, as before. There were 100 users of each type. The percentage of URLs in a given user's history drawn from the list of their interest area was varied. The results of this experiment are shown in Figure 4. The graph shows (as above) that as the user's browsing history becomes more representative of their interests, the number of good recommendations per user increases. It also shows that although there was significant overlap in noise across the two groups of users, the system was able to make good recommendations after each of the users' browsing history became sufficiently representative of their interests.

The above two experiments show that the system is successful in performing recommendations in a controlled setting, like those described above. This gives us a good indication that these results

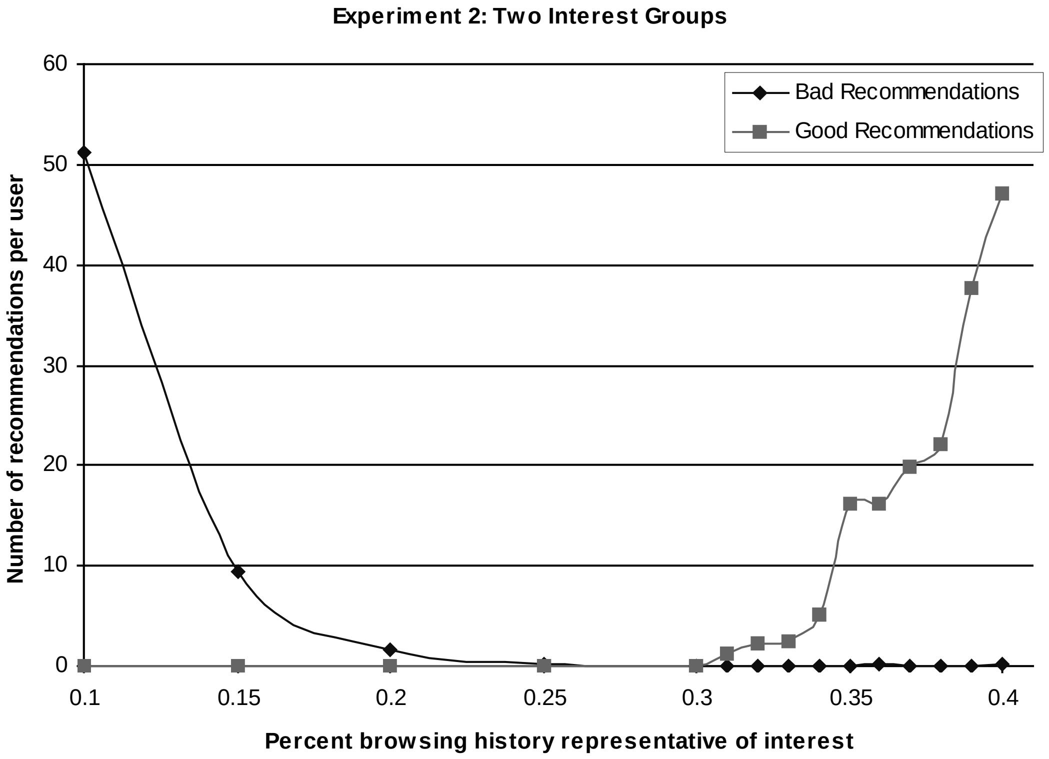


Figure 4: Results of experiment 2. The number of good suggestions per user increases as their browsing history becomes more representative of their interests.

will be consistent with the performance of the system once it is deployed. However, an empirical user study is in progress to verify the results presented here in a more naturalistic setting, over longer periods. Moreover, we are working on performing a study that compares our system with other collaborative recommender systems.

5.1 Interestingness And Relativeness

During the evaluation, we ran into a problem of how to decide whether a recommendation is related to a user's interests. While a user may visit a certain page, that page may or may not actually be of interest. While this is a valid concern, we argue that this would not be an issue. Remember, association rules capture the dependency among items, and if a set of URLs is read by a large group of users, there must be some relationship among this set of URLs. That is the underlying foundation of statistics. Our evaluation has shown that as users read more about a particular topic, the system's performance gets better. while we can not as yet prove our assumption true in a formal way, we are confident that applying association rules to navigation history would provide good recommendation.

6. RELATED WORK

Since the introduction of association rules, many algorithms have been proposed to improve their performance [1, 2, 4, 5]. While our work builds heavily on previous work, our research has its root in navigation history tracking and information recommender systems. Although not many systems have been built for navigation history collection, many research systems for recommending information have been built using various techniques. Most recently, the collaborative recommender systems have gained much attention. Among these systems are Alexa, GroupLens and Ringo.

Alexa. SurfLen has very similar system architecture and user interface as Alexa [3]. Unfortunately, there is no documentation about the working mechanism of Alexa. We can just guess from its help file. We surmise that the underlying mechanism of SurfLen is different from Alexa. Alexa's recommendation is largely based on a user's rating and explicit recommendations, while SurfLen gets its recommendation knowledge from mining users' navigation histories. We believe SurfLen's method is more natural to users. Furthermore, we make use of navigation history for other usage. One of the examples is community building on the Internet.

GroupLens. GroupLens [7] is a recommender system in the Usenet news domain that uses collaborative filtering. It requires users to enter ratings and then uses those ratings in two ways. First, it determines similarity between users using Pearson *r* correlation coefficients. Then it predicts how well users will like new articles based on ratings from similar users.

Ringo. Ringo [9] makes personalized recommendations for music albums and artists. The system maintains a dynamic profile of each user's interests (a list of items of previously liked or disliked), and then it computes the similarity of profiles using Pearson *r* correlation coefficient. Finally, it considers a set of the most similar profiles, and uses information contained in them to recommend items to the users.

Our research differs from these systems in that our underlying recommender mechanism is different. Mining association rules has provided several advantages. First, it does not require the user to explicitly rate an article, thereby making it is easier to use. Second, it offers a more robust way to make recommendations. Mining association rules allow us not only to group a set of related items, but also to group a set of close users.

7. FUTURE WORK AND CONCLUSION

We are doing comparison studies of performance between conventional collaborative information filtering method and methods based on data mining. We are also exploring ways to combine these two. In fact, some of our research on association rules has investigated the quality-based mining of association rules.

A common problem facing collaborative information recommender systems is "system bootstrapping". A collaborative filtering system requires a large group of users who have overlapping interests and that these users have interacted with the system for some time. The system will not be useful if it is sparsely used. Even if two users have agreed often on similar items, these two would not necessarily end up being nearest neighbors. This makes it difficult for the system to start making recommendations.

This problem can be alleviated by integrating other techniques. For example, we can combine collaborative recommendation with techniques for recommending based on content analysis, to find content-based connections between information items. For example, the system could keep a content representation for each of the documents encountered, and compute document similarity using the cosine measure [8]. In addition, the system could use statistical clustering techniques based on these content vectors to represent groups of similar documents as a single unit when computing association rules as above. Alternately, URL clustering based on heuristic URL similarity could be employed to help reduce the number of URLs considered. We can also embed our mining of a user's browsing history into a centralized system such as a search engine.

The implicit use of browsing information also brings up several issues with respect to the privacy of users. Further exploration is necessary to determine the severity of these issues and their effects on the system's use.

Information recommender systems extend access to information to more people. Collaborative recommender systems allow users to form virtual communities in which human knowledge can be reused automatically. Mining association rules provides a sound foundation for such systems. SurfLen allows people to share their Web browsing experiences to the mutual benefit of all of the users. SurfLen is the first step in an effort to apply association rule mining to real world problems.

8. ACKNOWLEDGMENTS

We would like to thank Julie Dubiner and David Wilson for their valuable suggestions. We are also grateful to three anonymous reviewers for their constructive comments and suggestions.

9. REFERENCES

[1] Agrawal, R. Imielinski, T. and Swami A. Mining association rules between sets of items in large databases. In Proceedings of the ACM SIGMOD Conference on Management of Data(Washington D.C., 1993), 207-216.

[2] Agrawal, R. and Srikant R. Fast Algorithm for Mining Association Rules. In Proceedings of the 20th VLDB Conference (Santiago, Chile, 1994).

[3] Alexa. Available at http://www.alexa.com.

[4] Fukuda, T., Morimoto Y., Morishita, S., and Tokuyama T. Data mining using two-dimensional optimized association rules for numeric data: scheme, algorithms, visualization. In Proceedings of the ACM SIGMOD International Conference on Management of Data (Montreal, Canada, 1996), 13-23.

[5] Holsheimer, M., Kersten M., Mannila, H., and Toivonent H. A perspective on database and data mining. In Proceedings of the First International Conference on Knowledge Discovery and Data Mining (Montreal, Canada, 1996).

[6] Pattie M. Agents that Reduce Work and Information Overload. Software Agents (eds. Jeffrey M.B.). AAAI Press, Menlo Park CA, 1997.

[7] Resnick P., Neophytos, I., Miteth, S., Bergstrom, P., and Riedl, J. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In Proceedings of CSCW94: Conference on Computer Supported Cooperative Work (Chapel Hill NC, 1994), Addison-Wesley, 175-186.

[8] Salton, G., Wong, A., and Yang, C. S. A vector space model for automatic indexing. Communications of the ACM 18(11), ACM Press, 613-620.

[9] Shardanand, U. and Maes, P. Social Information Filtering: Algorithms for Automating 'Word of Mouth'. In Proceedings of CHI95 (Denver CO, 1994), ACM Press, 210-217.

[10] Lycos. Available at http://www.lycos.com.

[11] Yahoo!. Available at http://www.yahoo.com

[12] Microsoft. Available at http://msdn.microsoft.com.