Learning Users Interests for Providing Relevant Information

Michal Chalamish and Sarit Kraus
Department of Computer Science
Bar-Ilan University, Ramat-Gan 52900 Israel
{sarit,merlich}@cs.biu.ac.il

In this paper we present a system that attempts to learn the surfing user's interests without asking him for any feedback. The system uses the user profile to conducts an independent search of the Internet, trying to find pages that the user have not read yet and might find interesting. The system consists of 3 modules: (1) The data-collecting module, which collects the data to be processed into the user's profile. The collected data is URLs the user reads and e-mail messages the user authorizes the system to process. (2) The user profiling module, which processes the collected data into a user profile, using the clustering method Dynamic Suffix Tree Clustering (DSTC). (3) The search module, which explores the Internet, searching for web sites the profiled user might be interested in.

1. Introduction

Most users of the Internet, experience both the excitement of the boundless information on the Web and the frustration of trying to actually find documents of interest [Kehoe and Pitkow 1999].

Our goal is to create an agent that will help the user to find what he is looking for, based on the user's interests without asking him explicitly what his interest are. Software agents seem to be a good solution to the problem of building a user profile [Edwards et al. 1996; Roure et al. 1996; Sheth 1994]. The user-profiling agent can learn from past experience. Actually, it can learn from the list of documents that have been accessed by the user and deduce an appropriate individual user profile.

Personalized information systems typically rely on the user supplying his profile, which is then used by an application to provide personalized services. Personalized newspapers and catalogues, are some examples of such services [Ardissono, Console and Torre 2001 ; NewsRounds]. For this type of system to work effectively, the user is required to specify keywords or phrases that represent his interests. The user needs to keep on updating his interests since there is no other way to detect changes of interest.

There is no question that user profiles are important, but furthermore that these profiles should be acquired automatically with minimal user input. After all, what use is a software agent that constantly requires the user to instruct it? By building an automated profiler agent, using the Dynamic Suffix Tree Clustering algorithm specified below, we attempt to create user profiles to provide the users with the information available on the Internet relevant to them.

In section 2 we will review different user-profiling agents. In section 3 we will detail the design of the various modules comprising the different components of our user profiling and search agent. We will also introduce the Suffix Tree Clustering algorithm [Zamir 1999]. The results of the experiments conducted on a number of users and an evaluation of the agent's performances will be presented in section 4.

2. Related Work

Soltysaik and Crabtree (1998) from BT used a standard information retrieval approach based on keyword vectors that are extracted from the user's e-mails and the WWW pages he browsed to construct the user profile. These vectors were used as the basis for grouping the data into clusters, the intention being that a sufficiently large cluster will represent a user's interest. Their profiling method used the clustering algorithm in two levels. The first level, called the *primary learner*, generates a set of interest clusters, while the second level, called the *secondary learner*, performs the classification of the interest clusters.

A different approach was taken by the system called Syskill & Webert [Billsus and Pazzani 1997]. Syskill & Webert learns a separate profile for each interesting subject of each user. Associated with each such subject is a URL called an *index* page. The index page is a manually constructed page that typically contains a hundred or more links to information providers. Syskill & Webert allows a user to explore the Web using the index page as a starting point. The user is required to rate each page he reads. It uses these rankings to build the user's profile and uses this profile to suggest other relevant pages accessible from the index page.

Billsus and Pazzani (1999) have also developed an intelligent agent named News Dude, which is designed to compile a news program for

Work presented in this paper has been conducted in part in the context of the EU-funded project IMAGEN IST-1999-13123. Project home page: http://www.imagenweb.org/

individual users. Based on feedback from the user, the system automatically adapts to the user's preferences and interests. The system's goal is to learn the user's interests in order to read news stories to him. A multi-strategy learning approach that learns two separate user-models was developed. One represents the user's *short-term* interests; the other represents the user's *long-term* interests. If a story cannot be classified with the short-term model, the long-term model is used. If the long-term model decides that the story does not contain sufficient evidence to be classified, a default score is assigned.

	User input	Specific application/ topic	Internet Search	Data rep. technique
Ringo	R	Specific	-	B
Letizia	N	Not Specific	-	B
Personal WebWatcher	N	Not Specific	-	---------
BT	R	Not Specific	-	P
Syskill & Webert	R	Not Specific	-	B
News Dude	R	Specific	-	B
ELFI	N	Specific	-	B
Our System	N	Not Specific	+	P

Table 1: *A comparison with other systems. In the second column R stands for "required" and N for "not required". In the fourth column, - stands for "not conducted" and + for "conducted". In the last column, B stands for "Boolean Vector" and P stands for "Phrases".*

Table 1 presents a comparison between our system, the ones described above and a few other well known systems including Ringo [Shardanand and Maes 1995], Personalized WebWatcher [Malednic 1996], Letizia [Lieberman 1995], and ELFI [Schwab, Pohl and Koychev 2000].

The main advantage of our system is that it does not require the user's input, it is not domain specific, it uses phrases rather than keywords in the profiling process and uses the user's profile to conduct an Internet search.

3. The User Profile and its Usage

Our system consists of 3 modules: the *data collecting* module which collects the data on the user, the *user profiling* module which processes the collected data into a user profile, and the *search* module which explores the internet, searching for web sites the profiled user might be interested in.

The most innovative module is the user-modeling module. It tries to identify the user's interests. An *interest* is a subject, which the user finds interesting. In our context, a subject will be considered an interest if it is discussed in a Web page the user reads and/or searches for or in an e-mail received or sent by the user. We assume that as the number of documents associated with an interest increases, the importance of the interest also increases. Thus, the user-profiling module will divide the documents into clusters. Each cluster will represent one of the user's interests. For the clustering process we apply the clustering Dynamic Suffix Tree Clustering (DSTC) method.

The system we present here is part of the IMAGEN (Intelligent Multimedia Application GENerator) project's Content Manager (CM). More details about IMAGEN and about its Layout Manager, which gets the CM's output as its input and displays it to the user, can be found in – [Brandmeier at al 2002 ;Rist at al 2002].

3.1. Suffix Tree Clustering

Suffix Tree Clustering (STC) [Zamir and Etzioni 1998; Zamir 1999] is a novel clustering algorithm designed to meet the requirements of post-retrieval clustering of Web search results. Most IR methods rely on the fact that the entire corpus is known prior to the processing of a given set of documents. In our case, as in Zamir and Etzioni's case the corpus is unknown and changes constantly. The system can never know which of the many existing Web pages will be the next one the user will read. As was demonstrated in [Zamir and Etzioni 1998; Zamir 1999], STC is the most compatible and efficient clustering algorithm in such situations.

STC is unique in treating a document as a string, not simply a set of words, thus making use of proximity information between words by clustering according to phrases. STC relies on a *suffix tree* to efficiently identify sets of documents that share common phrases. A suffix tree of a string is simply a compact tree of all the suffixes of that string [Zamir 1999]. *Suffix* in this context, is a string and all its sub strings. For example, given the string – "I know you know I know", its suffixes are: (1)"I know you know I know", (2)"know you know I know", (3)"you know I know", (4)"know I know", (5)"I know" and (6)"know".

A *suffix tree T* for an *m*-word string *S* is a rooted directed tree with exactly *m* leaves numbered 1 to *m*. Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty sub-string of words of *S*. No two edges out of a node can have edge labels beginning with the same word. The key feature of the suffix tree is that for any leaf *i*, the concatenation of the edge

labels on the path from the root to leaf *i* exactly spells out the suffix of *S* that starts at position *i*, that is, it spells out *S*[*i*..*m*]. The *label of a node* in the tree is defined as the concatenation, in order, of the sub-strings labeling the edges of the path from the root to that node. Figure 1 shows an example of a suffix tree. In a similar manner, a suffix tree of a set of strings, called a generalized suffix tree, is a compact tree of all the suffixes of all the strings in the set [Zamir 1999].

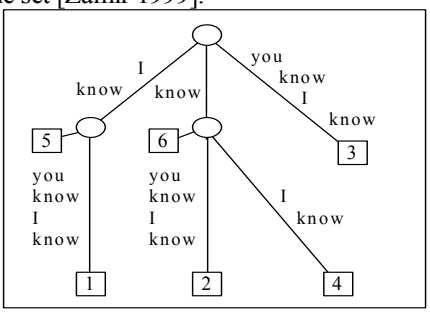

Figure 1: *An example of a suffix tree of the string "I know you know I know".*

A *generalized suffix tree T* for a set *S* of *n* strings S_k, each of length m_k, is a rooted directed tree with exactly $m_1+\ldots+m_k$ leaves marked by a pair of numbers (*k*,*l*) where *k* ranges from 1 to *n* and given k, *l* ranges from 1 to m_k. Each internal node, other than the root, has at least two children and each edge is labeled with a string of words of *S*. No two edges out of a node can have edge labels beginning with the same word. For any leaf (*i*,*j*), the concatenation of the edge labels on the path from the root to leaf (*i*,*j*) exactly spells out the suffix of *Si* that starts at position *j*, that is it spells out $Si\ [j..m_i]$.

Figure 2 demonstrates a generalized suffix tree of three strings "cat ate cheese", "mouse ate cheese too" and "cat ate mouse too". There are 11 leaves in this example (the sum of the lengths of all the strings) drawn as rectangles. The first number in each rectangle indicates the string from which that suffix originated; the second number represents the position in that string where the suffix starts.

3.2. The data collecting module

The *data collecting* module collects the data to be processed by the user-profiling module. It collects the user's e-mail messages and the WWW pages he read. The collected data will be referred to as the documents set. The data-collecting module operates without the user noticing it, reading and processing the Web pages the user reads. It uses only Web pages the user read for over 3 seconds, assuming that it is the minimum time needed for the user to decide whether the page interests him or not. In order to give the user a maximum feeling of security, our system allows the user to indicate that certain messages are not to be used while other messages aren't discreet and can be processed by the module.

Originally, STC uses as input not the full documents' text but rather the snippets returned by a search engine. In order to adapt the STC technique to general documents, the module extracts a number of the most important sentences for every document – we will refer to these as highlights.

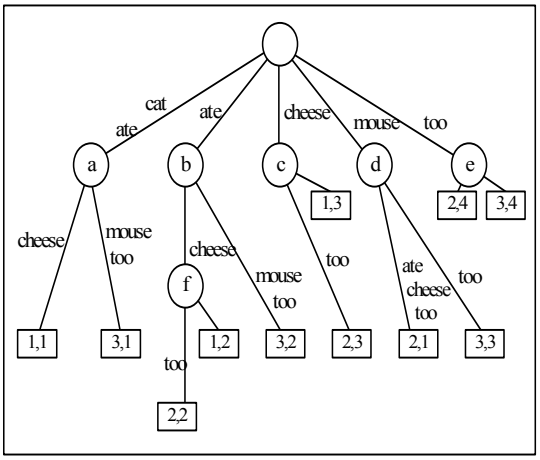

Figure 2: *The generalized suffix tree of the three strings "cat ate cheese" "mouse ate cheese too" and "cat ate mouse too".*

The collection of highlights drawn for all the documents, and the URLs of the Web pages, is the output of this module and the data to be processed by the user-profiling module. Currently, our system uses the 'Extractor' software to extract the documents' highlights (http://extractor.iit.nrc.ca/). The Extractor finds keyphrases based on a 10 steps algorithm [Turney 1999]. The algorithm searches the most frequently appearing single words. Based on these words it lists a number of the most frequently appearing phrases. The final phrases list is screened by a series of tests, which deletes the irrelevant phrases, such as phrases that contain verbs or adjectives or phrases that belong to a phrase stop list. Then, it extracts key sentences (highlights) showing the keyphrases in context.

3.3. The user profiling module

The *user profiling* module is the heart of the entire system. It gets as input the documents' highlights drawn by the data-collecting module, and updates the user's profile for future use of the search module. In section 3.1 we explained that we chose to use STC because our system doesn't have a fixed corpus. However, while the corpus is

not fixed, each profile is associated with a specific user. The associated user is fixed, but his interests may change over time. In order to take into account the non-fixed corpus and the specific user with changing interests we introduced the Dynamic STC (DSTC). An illustration of the user profiling module appears in figure 3.

The main task that is performed in both STC and DSTC is the clustering of the relevant documents. These clusters are used in our system to identify the user's interests. There are additional two main tasks that are performed in DSTC and are not needed in STC: deleting old interests and finding key phrases that are associated with the user's interests to be used in the search for new interesting Web pages. For these tasks, the system maintains, in addition to the suffix tree, a Document Map in which information on the documents read by the user is kept and Stemming Map in which information about words appearing in the suffix tree is stored. We these discuss data structures and the three tasks below.

3.3.1. Determining the user's interests

The key feature of the suffix tree is that each internal node v represents a set of documents and a phrase that is common to all of them. The label v_p of an internal node is the common phrase. All the leaves in the sub-tree of v correspond to sentences that have suffixes that start with the phrase v_p. Therefore, the set of documents containing v_p can be determined from these leaves. Table 2 specifies the phrase clusters of the suffix tree in figure 2 assuming that each string belongs to a different document.

Node	*Phrase*	*Documents*
a	cat ate	1,3
b	ate	1,2,3
c	cheese	1,2
d	mouse	2,3
e	too	2,3
f	ate cheese	1,2

***Table 2**: Internal suffix tree nodes as phrase clusters. The six internal nodes from the example shown in Figure 2 and their corresponding phrase clusters*

As can be observed in the example, the same document can belong to more than one cluster. In order to identify the most important interests of the user, the phrase clusters are merged into larger clusters, based on the overlap of their document sets. For the merging our system uses the same binary similarity measure defined in STC:

A binary similarity measure between phrase clusters is defined based on the overlap of their document sets. Given two phrase clusters c_i and c_j, with sizes $|c_i|$ and $|c_j|$ respectively, and $|c_i \cap c_j|$ representing the number of documents common to both phrase clusters, the similarity $sim(c_i, c_j)$ of c_i and c_j is defined to be:

if $|c_i \cap c_j|/|c_i| > \alpha$ and $|c_i \cap c_j|/|c_j| > \alpha$ then $sim(c_i,c_j)=1$; otherwise $sim(c_i,c_j) = 0$, where $0<\alpha<1$ (the used α was 0.6).

In order to merge phrase clusters the phrase cluster graph, where nodes are phrase clusters, is constructed (see an example in Figure 4). Two nodes are connected iff the two phrase clusters associated with the nodes have a similarity of 1. A merged cluster is defined as a connected component in the phrase cluster graph. Each merged cluster contains the union of the documents of all its phrase clusters.

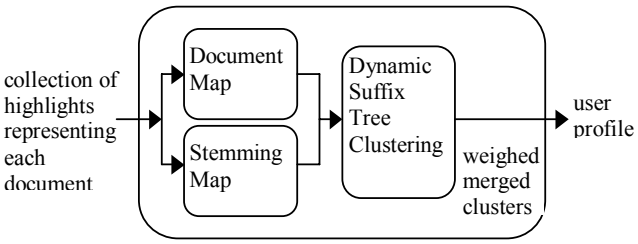

***Figure 3**: The User Profiling module. The documents' highlights are processed and saved in the Stemming Map and their URLs are processed and indexed in the Documents Map. After the highlights and URLs are processed, they are inserted into the suffix tree; phrase clusters are drawn, merged and weighed. The module outputs the weighed merged clusters as a user profile to be later used by the search module.*

The merged clusters are scored. The highest scored merged clusters represent the subjects the user is mostly interested in. The intuition behind the scoring algorithm is as follows:

- Merged clusters containing a large number of documents must have higher scores to those containing a small number of documents – we want the *dominating interests* to be identified.
- The newer merged clusters must have preference to the older ones - we want the profile to be *time adjusting*.

The first measure for the scoring is *NCD* - Number of Cluster Documents - representing the dominating interests factor. For every merged cluster i, NCD_i is the number of documents the merged cluster i includes. The second measure is *CDLD* – Cluster Documents read in the Last Days - representing the time adjusting factor. For every merged cluster i, $CDLD_i$ is the number of documents included in the merged cluster i that were read by the user in the last few days.

In order to normalize the score two more measurements are used. The first is *ND* – the Number of Documents in the Document Map. The

second is *LDD* – Last Days Documents - representing the number of documents in the Document Map entered by the user in the last few days (the current version uses three days).

For every merged cluster *i*: if LDD>0 then $weight_i = (CDLD_i / LDD)/(NCD_i/ND)$; otherwise $weight_i = -1$.

The next step is to sort the merged clusters according to their weight. This enables us to find the highest weighted interests for the searching phase. If two, or more, interests have the same score – the sorting is done according to their size. A number of such cases can occur:

1. All interests have a –1 score, meaning the user didn't surf at all in the last few days.
2. A number of interests have a 0 score, meaning no documents belonging to them were read in the last few days, but the user did surf.
3. A number of interests have the same score.

After the list is sorted we delete from it all the interests with fewer than three documents, since we consider them too small to indicate an interest.

The user-profiling module outputs the user profile, which is the user's interests, represented by the list of weighted merged clusters. Every merged cluster represents a single user interest, consisting of the different merged phrases and their documents.

3.3.2. Deleting old interests

Unlike the original STC where the built suffix tree is a temporary one and is built each time a Web search is performed, DSTC's tree should be maintained over time. In particular, highlights of new documents should be inserted to the tree very often (if the user surfed the Internet or authorized e-mail messages to be processed). However, the user's interests may change over time, and a subject that was of interest some time ago, may stop being of interest.

In addition, since STC was designed in order to process the results of a specific search, most of the documents inserted to it have a common phrase, and can be considered quite similar to one another. In our case, the topics of the documents may belong to many different domains – causing the resulting tree to be very broad. Thus, we must have a method of thinning the tree every pre-defined period of time.

For both reasons, there is a need to delete strings associated with old documents from the tree. To identify which documents are old and the parts of the tree that are associated with these old documents, the system maintains a map of the documents the user read and the dates of the reading.

Every document inserted to the tree is first indexed according to a documents map. This map saves for every Web document the document's URL and a list of the dates this page was read in the last month. This data is used for the deletion of the old documents from the tree as explained later.

In order to be able to delete documents from the tree, we first have to decide when a document is considered 'old' and needs to be removed from the tree. We consider a document to be 'old' if it was read more than *d* days ago (the used *d* was 30). If the interest which the document to be deleted belonged to, contains a lot of other 'newer' documents, then this is a *live* interest and continues to be of relevance, represented by the other documents. But, if the interest is not relevant anymore, by deleting the old documents everyday, we will finally delete all the documents representing it – and so – delete the interest itself. The deletion of old documents is done by, first, saving the last date the user entered this URL as part of the information saved in the tree leaves, and second, by going through the tree once a day and deleting every leaf or node that the document it represents is considered old.

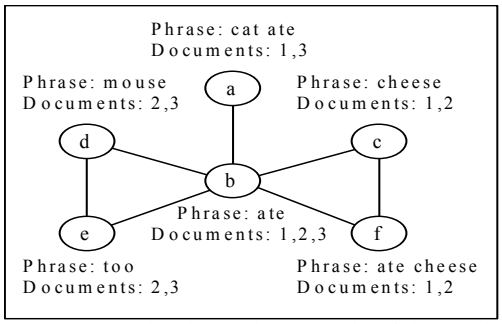

Figure 4*: The phrase cluster graphs of the six phrase clusters from Table 2. For $\alpha = 0.6$ there is a single connected component is the graph, representing one merged cluster.*

If the interest which the document to be deleted belonged to, contains a lot of other 'newer' documents, then this is a *live* interest and continues to be of relevance, represented by the other documents. But, if the interest is not relevant anymore, by deleting the old documents everyday, we will finally delete all the documents representing it – and so – delete the interest itself. The deletion of old documents is done by, first, saving the last date the user entered this URL as part of the information saved in the tree leaves,

and second, by going through the tree once a day and deleting every leaf or node that the document it represents is considered old.

There are a lot of URLs that change every day (i.e., news sites) or very often (i.e., informative sites with periodical updates). The URL does not change but the content of the page itself, and consequently its highlights, does. Since the URL doesn't change, the document will be mapped according to the Document Map to the same index, but the highlights inserted into the suffix tree will not necessarily refer to the same leaves. When the time comes to remove the first reading of this URL we want only the old readings to be deleted, but the newer readings to continue to provide the still relevant information. For that we need the reading date of every specific leaf to be part of the information it provides.

3.3.3. Identifying key-phrases

STC uses the Porter stemming algorithm [Porter, 1980]. The use of a stemming algorithm is the key for finding a phrase common to a number of documents. For example, the phrase "intelligent agents" and the phrase "intelligent agent" are overlapping phrases. If we will not use a stemming algorithm we will not identify them as such.

But the use of a stemming algorithm inflicts a problem later, in the Internet searching phase. Which word will we use when searching the Internet? The word 'dai' orienting in both 'day' and 'days', will not appear in this form in the documents the search engine will search, and if it will, will it really be relevant (DAI- Distributed Artificial Intelligence)? Both 'day' and 'days' will not necessarily give the desired result, since looking for one will not result in the other.

An *original word* is a word that appears in a highlight and needs to be stemmed. A *stemmed word* is the version of the original word, after it was processed by the stemming algorithm.

First, each stemmed word is mapped to all its original words; this way the system can later detect the original word(s) in the Stemming Map. This map saves the following information for every stemmed word: (1) the stemmed word (2) a list of all the original words, which after being processed by the stemming algorithm resulted in the stemmed word, and the number of times each appeared in the processed documents.

To solve the problem presented above, the system uses for the search query instead of every stemmed word, the two most frequent original words in its list. That way the stemmed word itself is not used, which is sometimes meaningless – like 'dai' in the example. Using the two most frequent original words provides as much relevant information as possible without enlarging the query too much, and making it meaningless.

For every original word, a list of all the documents' indexes the word appeared in and the number of times it appeared in each document is also saved. This list is needed for the updates of the Stemming Map when documents are deleted. Every time a document is removed from the suffix tree and from the Documents Map, all the words referring to it in the Stemming Map are updated as well. For example, both the words 'functions' and 'functionality' are stemmed to the root – 'function'. Let the word 'functions' appear twice in document 10 and three times in document 15. Let the word 'functionality' appear once in document 6 and five times in document 16. In this case, following the general format, the relevant entry in the Stemming Map will be:

function
functions 5 10 2 15 3
functionality 6 6 1 16 5

3.4. The search module

The *search* module is the module that searches the Internet looking for relevant Web pages the user will hopefully be interested in and screens the returned results.

Since it is impractical to take into consideration *all* the user's interests (there could be a dozen or more of them at any given time), the system considers only the three merged clusters with the highest weight. For each of the three interests we construct a query, send it to the search engines and screen the results.

In order to be efficient, we present the user, and therefore search for, up to 10 URLs as our search results – 4 for the first interest and 3 for each of the other two. This way we don't confuse the user nor flood him with information, but still try to save him a considerable amount of time searching the Internet.

The query for each of the interests (=merged clusters) is the concatenation of all the phrases the merged cluster represents. Between every two phrases in the query we add the relevant (according to the specific search engine) binary AND sign.

Now the system uses the information gathered in the word stemming and mapping phase. The two original words the stemmed word came from, which appeared most frequently in the documents, will replace every word in the concatenated query. This way the stemmed word (which may not have

a meaning by itself) is not used and not only one option of the original words for every stemmed word is used.

Finally, the relevant (according to the specific search engine) binary OR sign is added between every two original words replacing the same stemmed word. This step is done to make sure the search engine will not necessarily try to find both versions of the same word in the returned documents. If a certain stemmed word has only one original word, the OR sign phase is ignored.

Let us take for example an interest –

interest number: 3
documents: 3 21 25 20
weight: 2.07143
phrases: user profil profil librari automat

And let's presume the relevant entries in the Stemming Map are:

automat
automatic 5 21 2 25 3

profil
profiles 5 20 2 25 3
profile 2 21 2
profiling 3 6 2 9 1

user
user 13 18 2 20 6 21 2 25 3

librari
library 2 21 2 -1 –1
libraries 3 25 3

In this case, the query will be –

'(user profiles OR user profiling) AND (profiles OR profiling) AND (libraries OR library) AND automatic'.

We chose to use three search engines – Yahoo (at http://search.yahoo.com/), AltaVista (at http://www.altavista.com/) and Google (at http://www.google.com/).

Even though the search engines have their own rating functions we need our own method of deciding which document might interest the user and which might not. Our method has two parts. The first part dictates the cycle by which the different search engines results are considered. The second part is the result screening itself.

The query created in the former phase is now sent to every one of the three search engines to be searched. The first returned documents are the highest rated documents according to the search engines rating functions. To find the best documents, the first document returned by every search engine is considered, then the second and the third and so on – until the wanted number of documents the system is looking for is found.

Every merged cluster contains the list of the phrases that were merged from the original phrase clusters. These phrases are now used for the results screening.

First, the system draw highlights from the returned document. Next, the highlights are stemmed and last, the compatibility between the words in the document's highlights and the words in the merged cluster's list of phrases is computed. If at least 60% of the words in the document's highlights appear in the merged cluster's list of phrases it is considered interesting. But, if less than 60% of the words in the merged cluster's list of phrases appear in the document's highlights the document is considered not interesting, and the system moves on to inspect the next document.

After finding the wanted 10 documents, the only thing remaining is to send it to the Layout Manager in order to display it to the user.

4. Evaluations

In this section we discuss the experiments we conducted and their results.

The system was installed on a number of actual users. They have been asked to:

1. Activate the data collecting and user profiling modules. In the first week, nothing else was done in order to give the system time to create a preliminary user profile.
2. Starting in the second week, the experimenters were asked to send us the computed profile every three days. The computed profile has been processed by us into a list of the 10 highest weighing interests, each interest represented by all its phrases, and sent back to the experimenter for rating. Every interest has been rated by the user between 1- meaning 'not interesting' and 5 - meaning 'very interesting'.
3. Activate the search Internet module everyday, read the found URLs and rate them. Every URL has been rated between 1- meaning 'not interesting' and 5 - meaning 'very interesting', or 0 - meaning 'not entered'.

Since the experiments are done on actual users and have high and time consuming demands we can currently discuss only the results of 4 experimenters.

One of the experimenters read a lot of documents during the time of the experiment (over 500). At the beginning, his interests rating were low but they were rising constantly. However, his search results were rate quite low.

Another user has read a small number of documents (around 100) over the one and a half months of the experiment. He supplied us with

many results, giving the interests and search results high but inconstant ratings.

From the data we collected from the four experimenters can conclude that the system does improve its performances over time. The improvement is, of course, bounded since the ratings are finite, but we did expect and could see an improvement at least over the first two months the system has been activated. The system's performances depend inevitably on variants other than simply time. For example, the number of documents the user has read during this time period or the level of the user's interest focusing.

Low ratings of the search results could always be understood as not interesting enough. But we were surprised to find out that on a number of occasions experimenters gave high, and sometimes very high, ratings to the results of the search conducted for very low rated interests. It is difficult to explain the fact that there is no correlation between interests and search results ratings, but this in fact is the case.

5. Conclusions and Future Work

We described a system that learns the user's interests without asking him for any feedback and using it to search the Web. This was done using the DSTC that is a dynamic clustering mechanism that is appropriate for modeling a user's interests that are changing over time in the context of unknown corpus as the Internet. The preliminary experimental results are a reason for optimism regarding the performances of the system. We assume that farther system tuning is needed, but for that we must continue our experiments over a longer time period and have more participants' data to analyze.

6. References

Ardissono, L., Console, L. and Torre, I. (2001). An adaptive system for the personalized access to news. AI Communications, Vol 14(3), 129-147.

Billsus, D. and Pazzani, M. J. (1997). Learning and Revising User Profiles: The Identification of Interesting Web sites. Machine Learning 27,313-331.

Billsus, D. and Pazzani, M. J. (1999). Hybrid User Model for News Story Classification. Proc. of the seventh International Conference on User Modeling (UM'99), 99-108.

Brandmeier P., Kroener A. and Rist T. (2002). Layout Management for Cross-Platform Content Packaging. Proceedings of the 7th Annual Euromedia/WebTec Scientific Conference, 55-62.

Edwards, P., Bayer, D., Green, C. L. and Payne, T. R. (1996). Experience with learning agents which manage Internet-based-information. *AAAI Spring Symp. on Machine learning in IA*, Scotland.

Kehoe, C. and Pitkow, J. (1999) Tenth WWW Survey Report, The Graphics, Visualization & Usability Center, *Georgia Institute of Technology*.

Lieberman, H. (1995). Letizia: An Agent That Assists Web Browsing. Proc. of IJCAI-95, Montreal.

NewsRounds -Personalized news for the business of medicine. http://www.newsrounds.com/

Pazzani, M., Muramatsu, J. and Billsus, D. (1996). Syskill & Webert: Identifying interesting web sites. Proc. of AAAI-96.

Porter, M. (1980). An algorithm for suffix stripping. Program, 14(3):130-137.

Roure, D. D., Hall, W., Davis, H. and Dale, J. (1996). Agents for distributed multimedia information management. The first international conference on the Practical Application of Intelligent Agents and Multi Agents Technology, 91-102.

Rist T., Kroener A. and Brandmeier P. (2002). Layout Adaptation in a Portal for Personalised Cross-Platform Content Packaging. Article published in this proceedings.

Schwab, I., Pohl, W. and Koychev I. (2000). Learning to Recommend from Positive Evidence. *Proceedings of Intelligent User Interfaces*, New Orleans, Louisiana, 241-245.

Shardanand, U. and Maes, P. (1995). Social information filtering: Algorithms for automating "Word of Mouth". *ACM CHI'95 Mosaic of Creativity,* 210-217.

Sheth, B. D. (1994). A Learning Approach to Personalized Information Filtering. Master's thesis, MIT Media Lab.

Soltysaik, S. J. and Crabtree, I. B. (1998). Automatic learning of user profiles - towards the personalization of agent services. BT Tech. J., 16(3):110-117.

Turney, P. D. (1999). Learning Algorithms for Keyphrase Extraction. Information Retrieval, 2(4):303-336.

Zamir, O. and Etzioni, O. (1998). Web document clustering: A feasibility demonstration. Proc. of the SIGIR'98, 46-54.

Zamir, O. (1999) Clustering Web Documents: A Phrase-Based Method for Grouping Search Engine Results, PhD thesis, Univ. of Washington.