TOPIC DETECTION AND TRACKING IN PERSONAL SEARCH HISTORY

BY

KAMAL KANT GUPTA

B.Tech., Indian Institute of Technology Guwahati, 2006

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2008

Urbana, Illinois

Adviser:

Assistant Professor ChengXiang Zhai

Abstract

This thesis describes a system for tracking and detecting topics in personal search history. In particular, we developed a time tracking tool that helps users in analyzing their time and discovering their activity patterns.

The system allows a user to specify interesting topics to monitor with a keyword description. The system would then keep track of the log and the time spent on each document and produce a time graph to show how much time has been spent on each topic to be monitored. The system can also detect new topics and potentially recommend relevant information about them to the user. This work has been integrated with the UCAIR Toolbar, a client side agent. Considering limited resources on the client side, we designed an efficient incremental algorithm for topic tracking and detection. Various unsupervised learning approaches have been considered to improve the accuracy in categorizing the user log into appropriate categories. Experiments show that our tool is effective in categorizing the documents into existing categories and detecting the new *useful* catgeories. Moreover, the quality of categorization improves over time as more and more log is available.

To Father and Mother.

Acknowledgements

I would like to sincerely thank Prof. ChengXiang Zhai for his guidance and insight throughout my research. It was my pleasure to work with him.

I would take this opportunity to thank my colleague Bin Tan as well for helping me out in my research.

I would also like to mention that I learnt a lot from the Computer Science department at University of Illinois and was grateful to be a part of it. Also, I would like to thank the department for offering me Sara and Abassi Fellowship to support my graduate studies during Aug 2006 - July 2007.

Finally, I am thankful to all colleagues and friends who made my stay at the University of Illinois a memorable and valuable experience.

Table of Contents

List of Tables

List of Figures

1 Introduction

Due to popularity and widespread use of search engines like Google and Yahoo, the search patterns of a user raises an opportunity for text mining to acquire useful knowledge. This information can be used not only to improve the search accuracy of future queries but also for recommending information to the user.

There have been a lot of efforts in exploiting user search history. Most of these efforts make use of past queries and click through information. However, user may not find all the clicked information to be equally relevant, i.e., some documents are more useful than others for a given query. But, to the best of our knowledge none of the existing work takes this into account. In [13], authors made an attempt on using only those past queries and their click through information that are most relevant to the given query by giving weights. But they fail to distinguish between the different clicked documents for a particular past query. Motivated by the above reasoning, we developed a mechanism to track time information that user spent on a clicked document and use this to not only track user interests but also to detect new user interests and recommend them to the user.

Time is precious. It can't be saved, replaced, recovered, expanded or contracted. It is the dimension in which changes take place. In today's world, where everyone is running out of time, it will be pleasant to have a software application that can track the time spent by the user and help him in not only analyzing but also discovering his activities pattern. For example - a user might want to know the time he spends in watching movies, football matches, reading daily news and doing his field work (say computer science). Moreover, may be he wants to maintain a good ratio of 20:80 between his leisure and work time. Also, he might be interested in exploring his activities where he spends his time unconsciously. Keeping this in mind, we developed a tool that can help user in maintaining a healthy ratio between his work and leisure time but also help him analyzing and detecting his activity patterns.

In this work, we implemented a Time tracking tool that tracks and detects different categories and their coverage in the search history. To protect user private data and avoid over-burdening the server, we kept everything on

the client side and integrated our work with UCAIR (User-centered Adaptive Information Retrieval) toolbar, which is a client side agent. We talk about the whole system in section 5 and discuss its integration with UCAIR in section 6.

But in order to accomplish the above task, we need to first collect the personal data. This can be done either explicitly by asking the user, the activity in which he is engaged in, from time to time or implicitly by extracting from the user search log. Explicit inquiry is more precise than implicit, but its drawback is that the user needs to give input, which involves nontrivial user efforts. Therefore users are usually reluctant to engage in explicit inquiries. Since implicit information is inferred from normal search activities, there is no burden on the user.

Also, after obtaining the log we need to categorize the data into multiple topics. The concept of categorization is not new and has many great real world applications like it allows us to automatically organize news stories by the events they discuss by finding story boundaries, tracking these stories and discovering when something new happens. Also, categorization can help in automatically organizing books, articles, journals, and magazines in the library. Similarly, it can allow us to re-organize the emails and group them by topic. Another great application is to help individuals track their time.

Traditionally, one (usually experts in the related domain) assigns to each document in the collection its class. But it will be nice to have a system which can classify data automatically. Moreover, idea of designating classes by experts usually don't grow, i.e., the number of classes is fixed but the real system should be able to discover the classes by itself. However, sometimes user may want to set explicitly particular set of classes. Keeping both aspects in mind, we designed a hybrid system that not only can take input topics from the user but also discover them from time to time.

Given the problem of classifying the documents into multiple categories, the trivial approach could have been simply count the number of times the category words or their synonyms appear in the browsed documents and calculate the relevance.

However, there are various problems with this basic approach. The major problems with this approach are:

1. Firstly, it is quite sensitive to background words such as HTML tags and variables.

2. Secondly, it doesn't handle the case if document cannot be classified into specified categories.

3. Thirdly, we have to define the classes manually and there is no mechanism to learn new categories automatically.

4. A document may talk about multiple topics and hence can fall in many categories. Depending on the types (and relative importance) of words, we have to choose the appropriate category.

5. Finally, the category words are also fixed, i.e., it will not try to learn and evolve the category words to improve the class model, which is important in practice.

To improve upon our basic implementation, we followed a more general approach of building a language model for each category using maximum likelihood estimate by maximizing the probability of a category given a document for all documents. However, it turns out that it doesn't work well if we have a lot of noise in the data and everything except the space around the specified classes is the noise in our case.

In stead of learning a discriminative classifier for each document, we instead consider the generative approach. We model each document as being generated as a mixture of many topics. Each topic is defined by a language model. Also, a separate background category is kept to handle noise. In addition to categorization task, we also look for new categories and label them. We looked into various approaches to label these new categories [9] and discuss in details in section 5.

The other major challenge to make this work on the client side is the limited resources - processor, hard disk, memory, network bandwidth. We develop techniques which would not over consume any of the available limited resources. Keeping all this in mind, we developed an online algorithm that will run EM algorithm only on the new data and store the obtained results in a smart way, that can be used in the next run. We chose to keep each execution as on-demand. However, for commercial purposes it will not be a bad idea to make it run periodically.

The rest of the thesis is organized as follows. Chapter 2 talks about the related work and the novelity of our problem. In Chapter 3, we formally defined the problem. In Chapter 4, we briefly discuss about the UCAIR toolbar. Chapter 5 discusses the time tool architecture and its implementation in detail. In Chapter 6, we discuss its implementation with the UCAIR toolbar. Experiments set up and the details about the results can be found in Chapter 7. Chapter 8 concludes the thesis with a summary and discussion of future research directions.

2 Related Work

There are mainly two types of categorization techniques in machine learning, namely, supervised learning and unsupervised learning. In supervised learning, one tries to learn a function from the training data. The task is to predict a class label of any valid input after having looked at the training examples. On the contrary, unsupervised learning doesn't have any training data and hence requires manually setting the label.

Various classifying techniques in supervised learning have been studied in the past. Neural Network (Multi-layer Perceptron), Support Vector Machines, k-Nearest Neighbors, Gaussian Mixture Model, Gaussian, Naive Bayes, Decision Tree and RBF classifiers are some of the example classifiers for supervised technique. On the unsupervised learning side, we have data clustering, Expectation-Maximization (EM) algorithm, self-organizing map as some of the examples.

Although no previous efforts have been made in this direction but one can relate this work to topic detection and tracking (TDT) in information retrieval. However, there are a lot of differences between the two which makes this work quite novel and interesting.

Topic detection and tracking has been studied in the past and is still a very active topic in information retrieval. The goal of TDT is to identify event-based topics and monitor them in various news streams. TDT constitutes of three main evaluation tasks - Segmentation, Tracking and Detection. In segmentation, continuous news stream of data is split into distinct stories for tracking and detection. In tracking, a system is given some initial seed stories and asked to track them for further stories on the same topic. In contrast, detection performs unsupervised clustering on the incoming data and detects topic without any initial hints or clues. Another evaluation task is Link Detection, which determines whether two randomly selected stories are about the same topic or not. However, unlike above, this core task is a component technology, i.e., it can be used to address each of the other tasks.

The TDT Pilot Study [1] ran from September 1996 through October 1997. This study corpus spans the period from July 1, 1994 to June 30, 1995 and includes nearly 16,000 stories, with about half taken from Reuters

newswire and half from CNN broadcast news transcripts. A set of 25 target events has been defined to support the TDT study effort consisting of both expected and unexpected events. Similar study was conducted in the year of 1998 [5], which consist of data collected from the first half of 1998 and taken from 6 sources including two newswires, 2 radio programs and 2 television programs. There were a total of 57 thousand stories including 630 hours of audio in this corpus.

There were many participants and they proposed different interesting methodologies to attack the above tasks. Most of them have relied on some sort of clustering technique like Single Pass Clustering [1, 17, 7] or hierarchical group average clustering [17]. Also, Hidden Markov Models [10], Rocchio [16], k-nearest neighbor [16], Naive Bayes [11], probabilistic Expectation-Maximization models [2] and Kullback-Leibler divergence [6] have been used. TDT research has continued open evaluation in TDT1999-2004.

In [8], authors proposed another approach for TDT that formalizes temporal expressions and evaluates the relevance of two spatial reference with respect to an ontology.

The above problem is very similar to ours in a way we also detect and track a user log but ours is a more difficult problem since we have to track the topics without any prior training, i.e., unsupervised learning.

In [4], authors developed a client side extension to email clients by grouping messages discussing the same topic and automatically labeling them to summarize the contents by clustering the emails. They used the single link clustering with a distance measure as *tf * idf* similarity measure and non-textual information like sender receiver relationships and behavior measures such as the percentage of replied emails, contact ranking based on email volume, past behavior and reply timing to either match it to one of the existing group or form a new topic.

3 Problem Formulation

The abstract problem that we are trying to solve is to categorize documents into existing topic categories and recommend new categories. Let, $\theta_1, \theta_2, ...\theta_k$ be the already known topics, where k is some constant and $\theta_{k+1}, \theta_{k+2}, ...\theta_m$ be the new topics added by the user, where m is some constant and suppose $\theta_{m+1}, \theta_{m+2}, ...\theta_p$ be the topics discovered during the current run, where $p \geq m$, $m \geq k$, $k \geq 0$. Let, θ_B be the background cluster.

Let, i be the i^{th} theme added by the user, where $k < i \leq m$ and,

$$W_i = w_{i1}, w_{i2}, ..., w_{ij} \tag{3.1}$$

be the synonym words and,

$$W = \bigcup_{y=1}^{m-k} W_y \tag{3.2}$$

be the union of all the synonym words of the new topics added during the current run.

Also, let $cluster_1, cluster_2, ..., cluster_{p-m}$ be the new clusters corresponding to discovered $\theta_m + 1, \theta_m + 2, ..., \theta_p$ topics and,

$$W_i' = w_{i1}', w_{i2}', ..., w_{ij}' \tag{3.3}$$

be the generated model words for the i^{th} cluster, where $m < i \leq p$ and,

$$S_{i1} = s_{i11}, s_{i12}, ..., s_{i1l} \tag{3.4}$$

be the synonym words for the first word of the i^{th} cluster and,

$$S_i = \bigcup_{y=1}^{j} S_{iy} \tag{3.5}$$

be the union of all the synonym words of all the words of the i^{th} cluster and,

$$S = \bigcup_{z=1}^{p-m} S_z \tag{3.6}$$

be the union of all S_i for the detected topics

Our goal is then, given a query and a set of documents visited by the user, we either need to categorize them into existing topics or form a new category and compute the time spent in each category. Formally,

$$\{(W, S, \{Q_1, Q_2, ..., Q_N\}, \{D_1, D_2, ..., D_N\})\} \longrightarrow (\theta_B, \theta_1, \theta 2, ..., \theta_k, \theta_{k+1}, .., \theta_m, \theta_{m+1}, ..., \theta_p) \tag{3.7}$$

where N is the new search log that have been never analysed before.

4 UCAIR Toolbar

UCAIR[1] is a client side agent and is like a Google toolbar[2] where user can submit the query and get the results directly on the browser. The main difference though between the two is that UCAIR is the client-side agent, that personalizes the search results and uses it to further improve the future queries by capturing the query history and the click through information. The information flow between the user and the search system without (top) and with (bottom) UCAIR Toolbar is shown in Figure 4.1.

Traditionally, user submits the query on search engine, the query is then processed at the server side usually based on popularity or relevance and the results are sent back to the client browser. The major problem with this system was that it is not adaptive to any particular user and the results returned by the search engine solely depends on the current context, i.e., for ambiguous queries like *jaguar*, everyone receives the same result, which was definitely not good. To better understand this, consider an example query "Jaguar" which can be *Panthera onca, a Cat*, *Mac OS 10.2*, *Jaguar wiki*, *a branded Car*. Without knowing any other information, search engine could best return the mixed bag of results from different meaning groups. This was definitely not optimal. In fact, the situation was even worse for queries like "Java" where one group dominate all the other less popular meanings. This problem is often referred to as "one size does not fit all" and is one of major bottlenecks of most existing search engines [12].

Therefore it was critical to identify different user specific needs and deliver personalized results specific to a user. The UCAIR thus added another layer in between the user and the search engine to collect extra information besides the query to put search in context. Specifically, the following changes had been made with the introduction of UCAIR.

1. The implicit feedback information (query and click through information) are captured to personalize and produce better results in ambiguous queries, which will update the user model. This user model can then better server the individual user need.

[1] Downloadable from http://sifaka.cs.uiuc.edu/ir/ucair
[2] http://toolbar.google.com

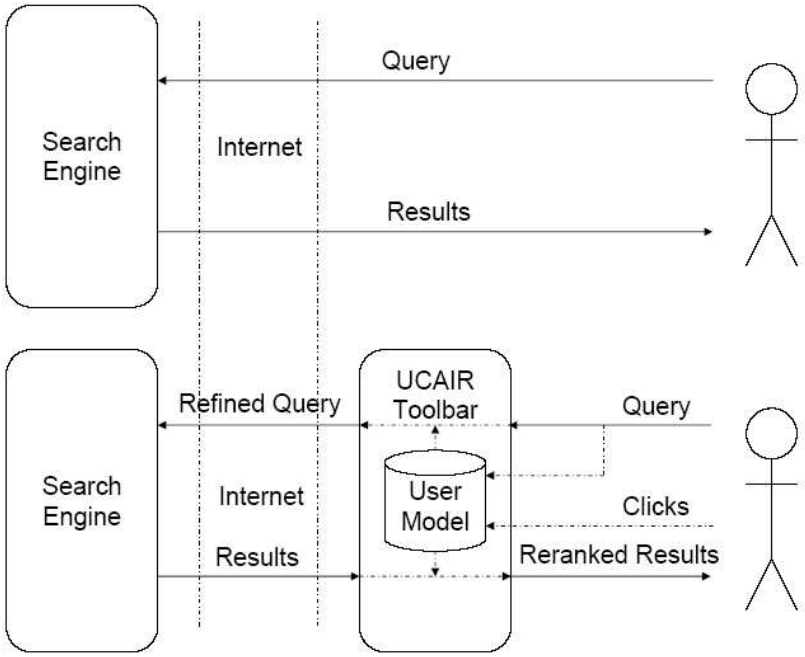

Figure 4.1: UCAIR Architecture

2. When user submits a query, UCAIR tries to expand the original query in accordance with the users search context, which will better serve the ambiguous query.

3. Upon receiving the result, UCAIR reorganizes the results as well (by re-ranking it according to the inferred user need). The goal is to making the top results most relevant or diversifying the top results to facilitate potential feedback.

Figure 4.2 shows the original mixed results with pages about Jaguar cars and Jaguar software. Figure 4.3 shows how UCAIR can re-rank search results from Google and optimize search results for a user searching information about the Jaguar car using the query "jaguar". It shows the automatically re-ranked results by UCAIR after the user has viewed the 1^{st} page, which is about Jaguar cars. The new results no longer have pages about the Jaguar software; instead, three new pages about Jaguar cars have been pushed up by UCAIR, which were originally ranked down in the results from Google.

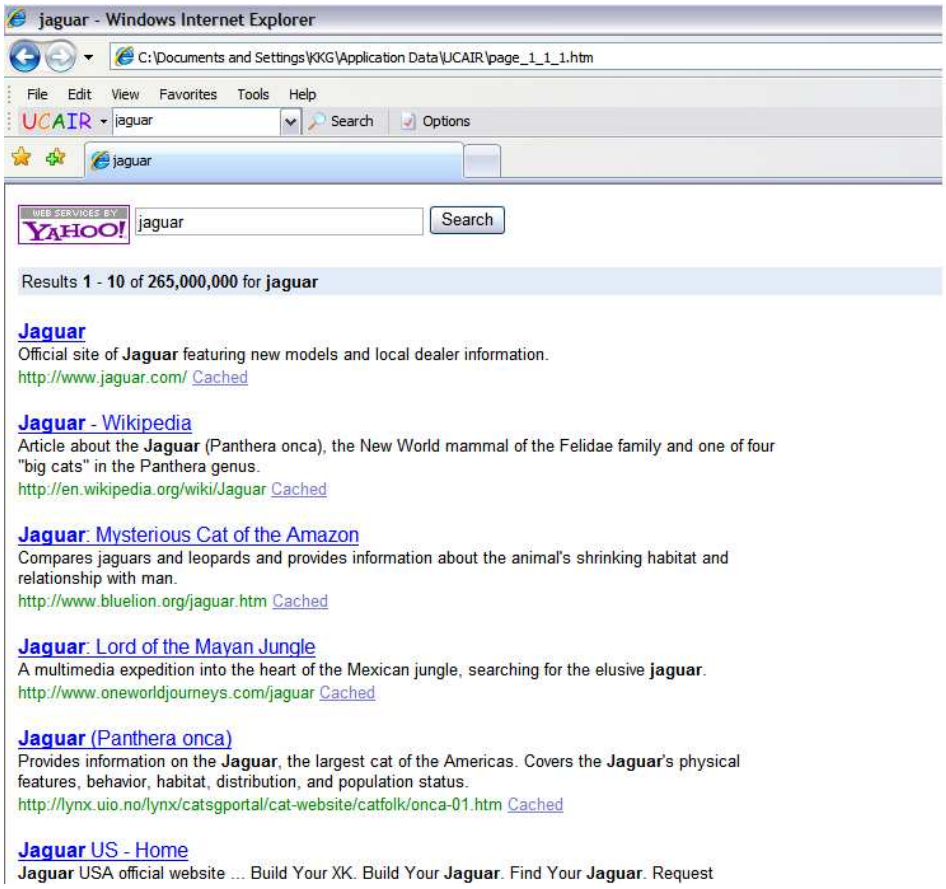

Figure 4.2: Search result without context

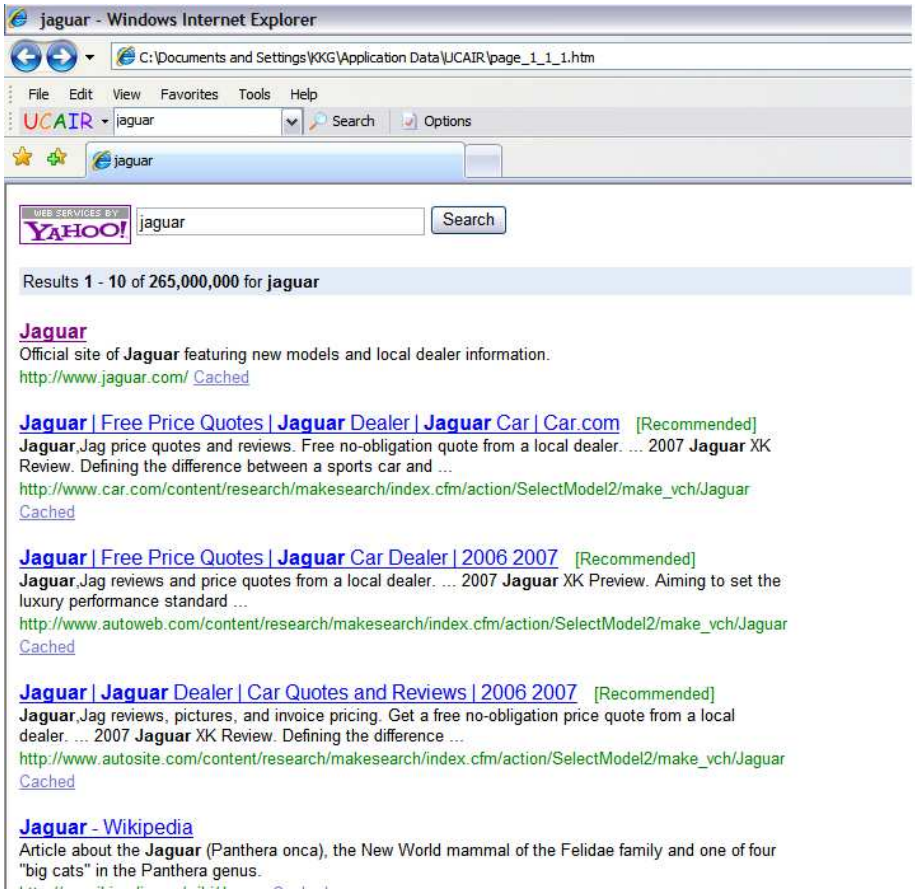

Figure 4.3: Reranked search page

5 System Design

The Time tool consists of four main actions:

1. Add Category - This will store the topic names specified in the search box in the temporary location that will be read when Time tool runs. If nothing is specified, it will report an error.

2. View Results - This will display the contents of the result file in the appropriate tabular form.

3. Reset TimeTool - This will simply delete all the stored Time tool files.

4. Run TimeTool - This is the core action of the system and constitutes the main algorithm. We will discuss this in detail below.

The system design can be divided into the following phases:

1. System initialization

2. Preparing parameter files

3. Retrieving synonym words

4. Data crawling

5. Indexing

6. Expectation-Maximization Step

7. Setting Labels

8. Processing results

9. Storing statistics

Figure 5.1 shows the system architecture and how the various parts are inter-connected.

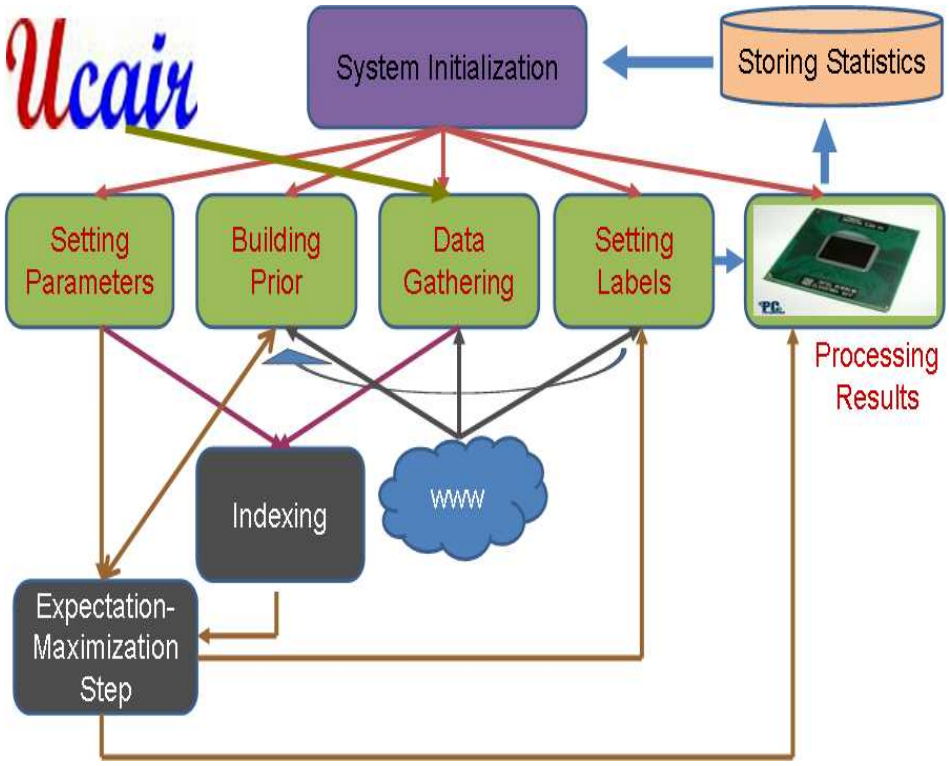

Figure 5.1: Time tool architecture

5.1 System Initialization

In this step, system gets initialized. If the tool has never been run or it has been just resetted then it will create and initialize the appropriate system files. Not only it will create the *Other*, a background category but also look for categories being added by the user using the Add category action.

Otherwise, it will look for old statistics and if found, it will dynamically load them and also look for new categories, if any.

We allow for additional categories to be detected by the system in addition to being specified by the user but in a limited way. We recommend one category per four categories specified by the user. This is done to make the system faster and to avoid interfering with the user routine too much. However, this feature can be very helpful since it might be able to discover a category that a user may not be too aware of.

5.2 Setting Parameters

After initializing the system and gathering the data, system initializes and creates all the parameter files required for indexing and Expectation-Maximization

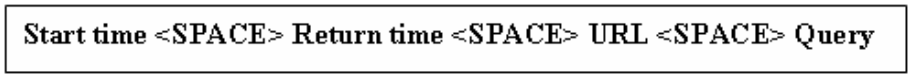

Figure 5.2: Time tool format

step with the appropriate parameters.

5.3 Building Prior

To get the best results with the Expectation-Maximization (EM) algorithm we need to provide some prior information to the EM algorithm by adding some similar words to our categories. In order to build such a rich prior file for the new categories, we retrieve synonym words from multiple online dictionary websites. The list of synonyms from each of these websites is then integrated to form a big list of synonym words. Since we don't have any prior information about the new category we assign equal probability to all of these synonym words in the prior file, sum of which is equal to 1.

The EM algorithm, as we will discuss below form multiple clusters equal to number of categories, and output the result along with high probability words for each category. These words come directly from the document, while generating the words in the document in the E-Step. For better categorization, our algorithm thus uses this information for the existing categories.

For the existing categories, we not only use the synonym words but also the cluster words from the previous run. As EM algorithm will gain more confidence in top words, the contribution of synonym words in the prior file will go less and less. This is like an adaptation. We initially gave some seed to EM algorithm to categorize documents and then use the EM algorithm output to further tune our prior file.

We chose top 100 words generated by the EM algorithm and discounted the sum of probabilities of those words, to assign new probability to each of the synonym words to keep the sum of probabilities equal to 1.

5.4 Data Gathering

UCAIR keeps the log of the user activity in the XML form. XML is the extensible markup language whose main purpose is to facilitate the sharing of structured data across different information systems [3].

UCAIR stores the query, start time, and internal ranking for each search. In addition, UCAIR also keeps a log of clicked time, URL of the page, title of

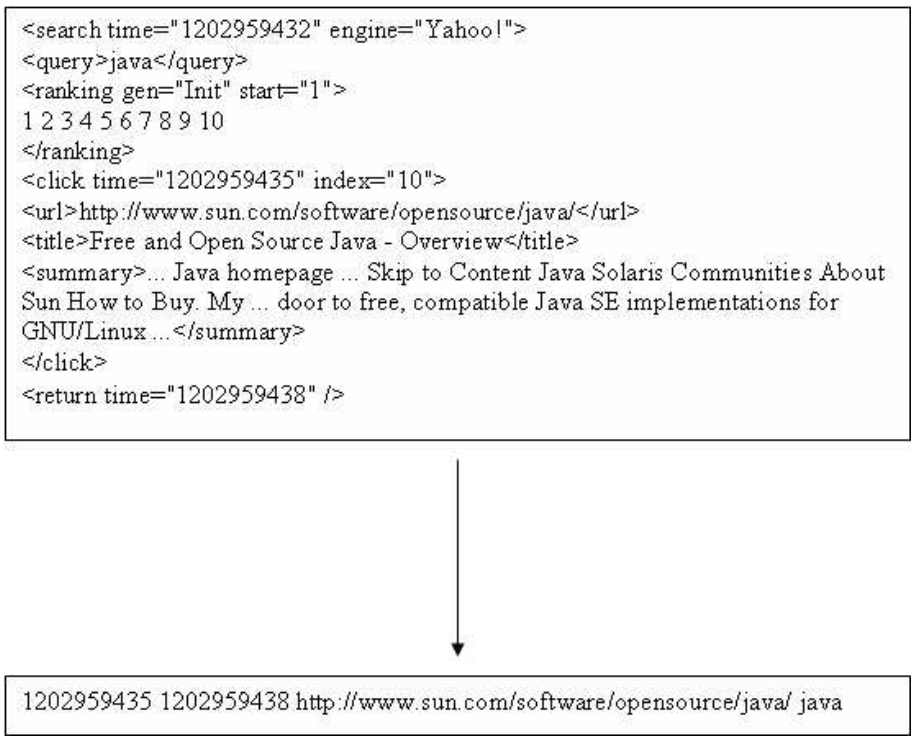

Figure 5.3: Mapping of UCAIR log to Time tool format

the page, summary snippet returned by the Yahoo or Google search engine, and the return time.

In order to keep our tool generic enough and store only required fields, we defined a different format to store the click time, return time, URL and title of the page. Hence, one only needs to write a function which can convert the log to Time tool accepted format to integrate our tool with any client side agent. Figure 5.2 shows the format of time tool. Figure 5.3 shows an example mapping of UCAIR log to our format.

Right now, system converts the whole log in to time tool format each time and keeps track of the position from where new log begins. It then parses and extracts the URL. The extracted URL is then downloaded at a specific location using our function which after opening the page, read it in a buffer and then save it in a temporary location. It is then later stored in the special format as required for indexing.

Also, given the start and the return time, time spent on each document can be easily calculated and associated with each document.

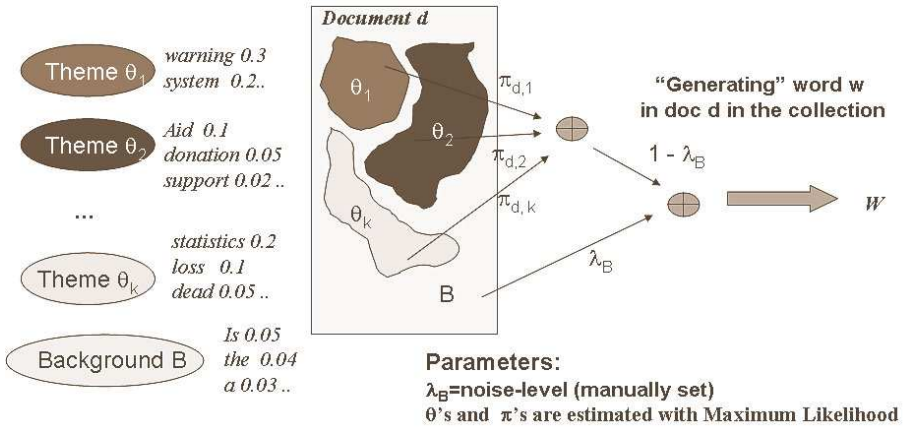

Figure 5.4: Probabilistic latent semantic analysis

5.5 Indexing

Once the articles are crawled, we build a DocumentManager and *KeyfileIncIndex* over the entire text for all the articles using the BuildDocMgr application of Lemur toolkit [14].

KeyfileIncIndex builds an index assigning term ids, doc ids, tracking locations of term within documents, and tracking terms within documents. It expects a DocumentProp to have the total number of terms that were in a document. Further, it expects that remove of stop words and stemming (if any) occurs before the term is passed in. If used with an existing index, it adds new documents incrementally. It stores the records in keyfile B-trees and provides index API to use the index.

5.6 Expectation-Maximization Step

Expectation-Maximization (EM) algorithm, as shown in figure 5.4, is used for finding maximum likelihood estimates of parameters in probabilistic models, where the model depends on unobserved latent variables. EM alternates between performing an expectation (E) step, which computes an expectation of the likelihood by including the latent variables as if they were observed, and maximization (M) step, which computes the maximum likelihood estimates of the parameters by maximizing the expected likelihood found on the E step. The parameters found on the M step are then used to begin another E step, and the process is repeated.

The EM procedure 5.5, then, is:

1. Initialize the distribution parameters

$$
\begin{aligned}
p(z_{d,w} = j) &= \frac{\pi_{d,j}^{(n)} p^{(n)}(w|\theta_j)}{\sum_{j'=1}^{k} \pi_{d,j'}^{(n)} p^{(n)}(w|\theta_{j'})} \\
p(z_{d,w} = B) &= \frac{\lambda_B p(w|\theta_B)}{\lambda_B p(w|\theta_B) + (1-\lambda_B)\sum_{j=1}^{k} \pi_{d,j}^{(n)} p^{(n)}(w|\theta_j)} \\
\pi_{d,j}^{(n+1)} &= \frac{\sum_{w \in V} c(w,d) p(z_{d,w} = j)}{\sum_{j'} \sum_{w \in V} c(w,d) p(z_{d,w} = j')} \\
p^{(n+1)}(w|\theta_j) &= \\
& \frac{\sum_{i=1}^{m} \sum_{d \in C_i} c(w,d)(1 - p(z_{d,w} = B)) p(z_{d,w} = j)}{\sum_{w' \in V} \sum_{i=1}^{m} \sum_{d \in C_i} c(w',d)(1 - p(z_{d,w'} = B)) p(z_{d,w'} = j)}
\end{aligned}
$$

Figure 5.5: Mathematical formulation of EM algorithm

2. Repeat until convergence:

   (a) E-Step: Estimate the *expected* value of the unknown variables, given the current parameter estimate, i.e., probability of word in document d being generated from cluster j (given by p($z_{d,w} = j$) as shown in 5.5)and from background (given by p($z_{d,w} = B$) as shown in 5.5)through the application of Bayes' rule.

   (b) M-Step: Re-estimate the distribution parameters to maximize the likelihood of the data, given the expected estimates of the unknown variables. The last two equations in 5.5 refers to M-step.

This step is the heart of the total system. In this step, we run EM algorithm on the new data, the result of which will later be integrated with the previous results.

5.7 Setting Labels

This section is about detecting new categories and as we discussed our tool recommends categories to the user to help him discover some unknown categories.

Specifically, in this step we label the new category that has been detected in EM step. A good label should be

1. understandable to the user,

2. should capture meaning of whole cluster words,

3. distinguish it from other clusters,

4. should not be too specific, for eg, Music will be preferred over *The Beatles*

Instead of simply labeling a category with the top words of the new cluster found in the previous step, we chose a more exhaustive approach, as presented in [9], of automatically labeling the new cluster. To label a new category, we have used the following factors:

1. Probability of the word from the topic word distribution from EM step should be high.

2. Label should correlate to the new category as much as possible and should be distinguishable from other categories. For this we calculated the Kullback-Leibler divergence between the topic word distribution (cluster words) and the label word distribution (similar words around label).

3. Generality of the label word by calculating the space of label word distribution. This is quite intuitive as more specific words will have much smaller space as compare to generic words.

Considering the above factors in mind, we propose a label score as a function of above three factors:

$$labelscore(x) = count * P(x) * \alpha + P(x) + KL * P(x) * ratio \tag{5.1}$$

Where,
label score(x) = final score of x
count = Similar words around a label word
P(x) = Probability of word in topic word distribution
α = constant value of 0.08
KL = Kullback-Leibler divergence between the topic and the label word distribution
ratio = constant value of 0.95
We chose the label such that

1. it has the maximum score and,

2. it is non-overlapping with the existing categories label.

Again, prior file will be updated from the synonym and the topic words for the newly discovered category.

5.8 Processing Results

In this step, the result file of the Expectation-Maximization step is parsed and the result of the current run is integrated with the previous results.

The actual time spent on a document may not be equal to $\sum_{alldocs}(return-clickedtime)$, because one may click the document and then go to a different page in a separate window and start reading that page. To account for this kind of situation, we decided to switch from linear function to a more conservative logarithmic function.

For documents where user spends a small amount of time, we know that log x $\simeq$ x and for documents where user spends a lot of time x $\gg$ log x and as x goes to ∞, log x also goes to ∞. Also, note that taking log wouldn't much affect the percentage of time spent in a category. Hence, their relative percentage will remain the same but now the absolute number can give a better indication of how important that category is for a user.

5.9 Storing Statistics

In this step, the system stores all the updated results and statistics to a specific location so that it can be retrieved in the next run.

6 Integration with UCAIR

Since a user log is very critical and contain personal information we have integrated our tool with the client side agent UCAIR to account for user privacy. First, let us try to understand the internals of UCAIR.

Each action on UCAIR toolbar is associated with the command handler defined in IRBar file and each command handler is assigned a unique number which is specified in Resource header file. On clicking a menu item on the toolbar, the corresponding command handler gets activated which calls its associated function defined in MainWindow header file. The definition of the corresponding function can be found in MainWindow class. Thus, to add menu items, we made appropriate changes in the IRBar, Resourse header, MainWindow header, MainWindow class files.

For simple tasks like resetting the time tool or viewing the results, we wrote our implementation in the MainWindow class itself with the help of global functions defined in Global class. For more complicated actions like adding a category or running the time tool we need to queue our request which is discussed below.

The UCAIR system works by calling the appropriate system function on each user action. Now each user action and its corresponding system action are represented by the class. Instead of simply enumerating user action and representing system action by functions, this has been done for many reasons like, adding modularity. However, the main reasons were to be able to pass the attribute value associated with user actions (like adding a new category to Time tool or submitting a new query). Also, system actions may be asynchronous (e.g. while downloading one has to wait for the response) and internal state needs to be maintained. Because of all the above reasons, UCAIR implementers chose to represent them with classes.

Both user actions and system actions are subclasses of Action class which mainly defines the GetName() and Execute functions. In user action class, we defined GetName function and in our System action we redefined functions like GetName, Execute (from Action superclass). Please refer to Figure 6.1 for the internal structure of UCAIR. UA refers to User Action and SA refers to System Action in the figure and dotted lines signify *Callback* class,

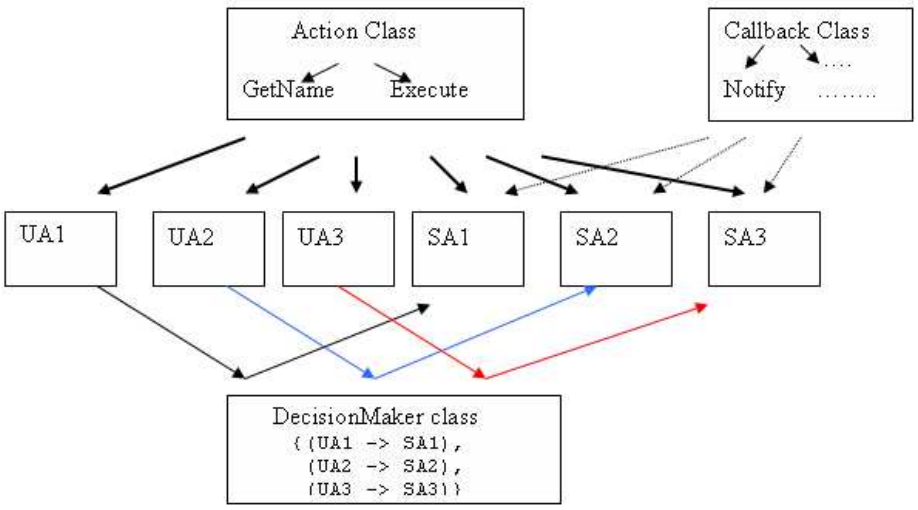

Figure 6.1: UCAIR internal structure

which may or may not be subclassed by system actions classes.

The other thing we updated is DecisionMaker class where we made an entry in the constructor, which will link the user and the corresponding system action through the string names, and we handled the *if case* in NewAction function as well (which follows a strategy pattern). Very briefly, this pattern has been used to be able to select different algorithms at runtime. For more details please refer to [15].

Finally, we created an instance of new user action class and called respond function of Decision Maker class from our handler function defined in MainWindow class, which will queue requests and automatically call execute function on the corresponding new system action class defined by us. For other UCAIR specific user and system functions, please refer to [12].

The *Time Tool* can be downloaded along with UCAIR from the following URL: `http://sifaka.cs.uiuc.edu/ir/proj/ucair/download.html/`

Figure 6.2 shows the new items that are added to the UCAIR toolbar.

1. Add Category
2. View Results
3. Run TimeTool
4. Reset TimeTool

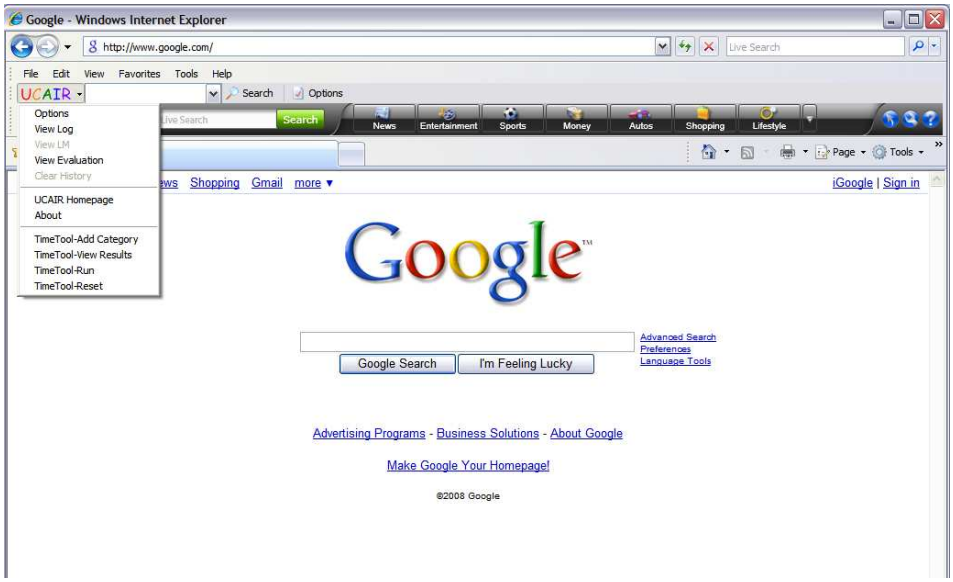

Figure 6.2: Time Tool

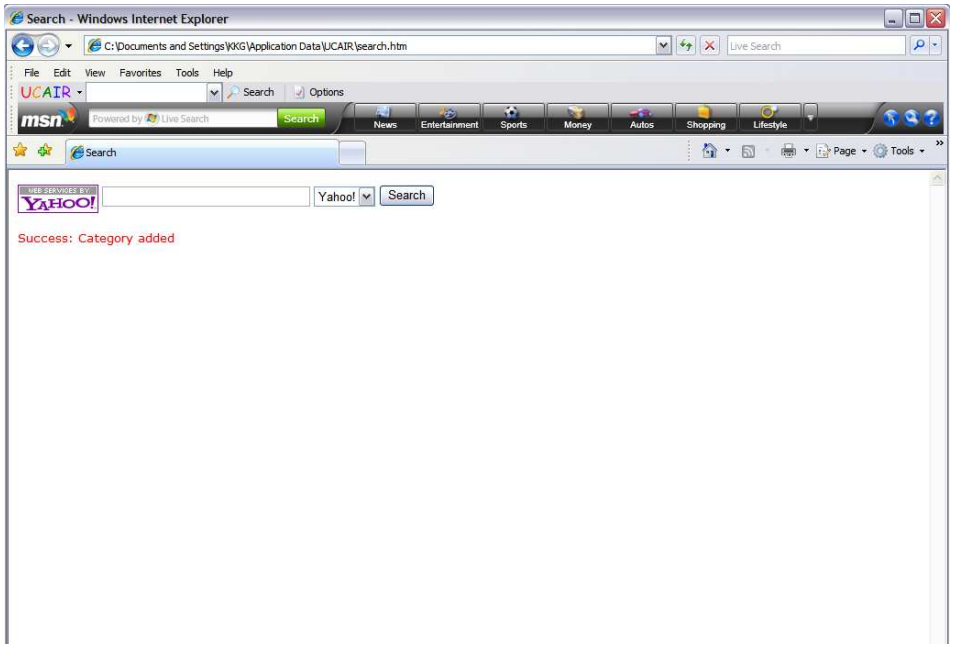

Figure 6.3: Success Page on adding a category

6.1 Add Category

This is the tab to add a category to the existing set of categories. Success message will be displayed upon successfully entering the category. However, if user forgot to specify any category in the search box, an error message saying "Please enter the category" will be displayed. Figure 6.3 shows that *Movie* category being added successfully. Figure 6.4 shows the error page, in case nothing was input.

6.2 View Results

A user can see the graph of his activity at any moment by clicking this menu item. Figure 6.5 shows how user spends his time in Movies, Sports, Music, Other. Where, "Other" refers to the background cluster. The number in front of each category gives the estimate of the time spent in each category in percentage. New labels, if any, would appear at the end of this table.

6.3 Run TimeTool

Figure 6.6 shows the snapshot of the page showing the progress of each step. A user can run the time tool in an adhoc way.

6.4 Reset TimeTool

A user may want to delete his past records, and start afresh. Figure 6.7 shows the page that will be displayed on success. All the previous results and categories will be lost on clicking this. The error dialog box will pop up if some files were missing or system was already reset.

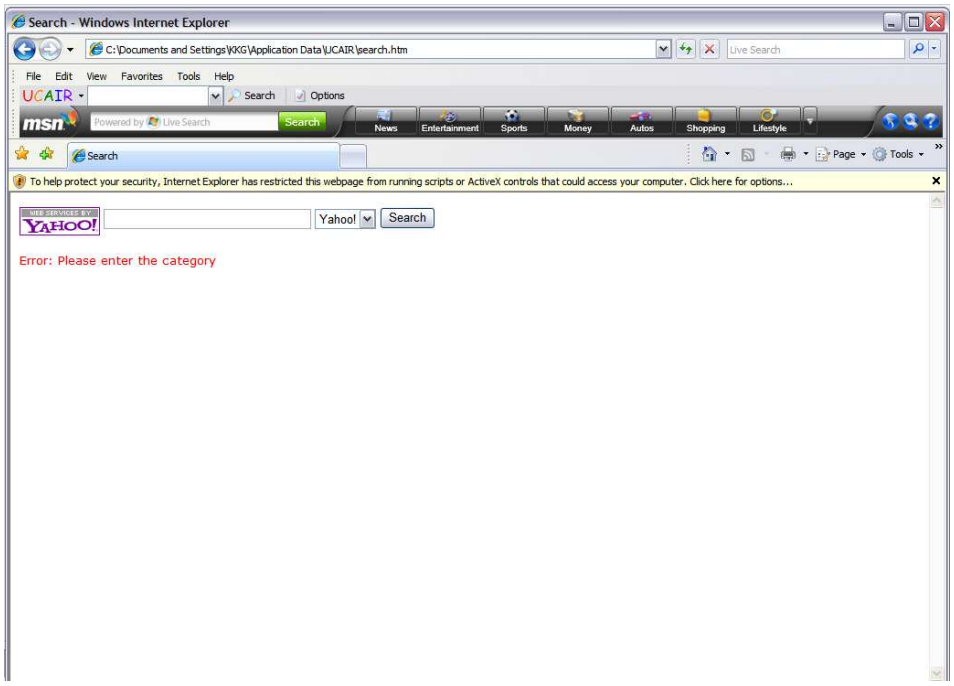

Figure 6.4: Error page

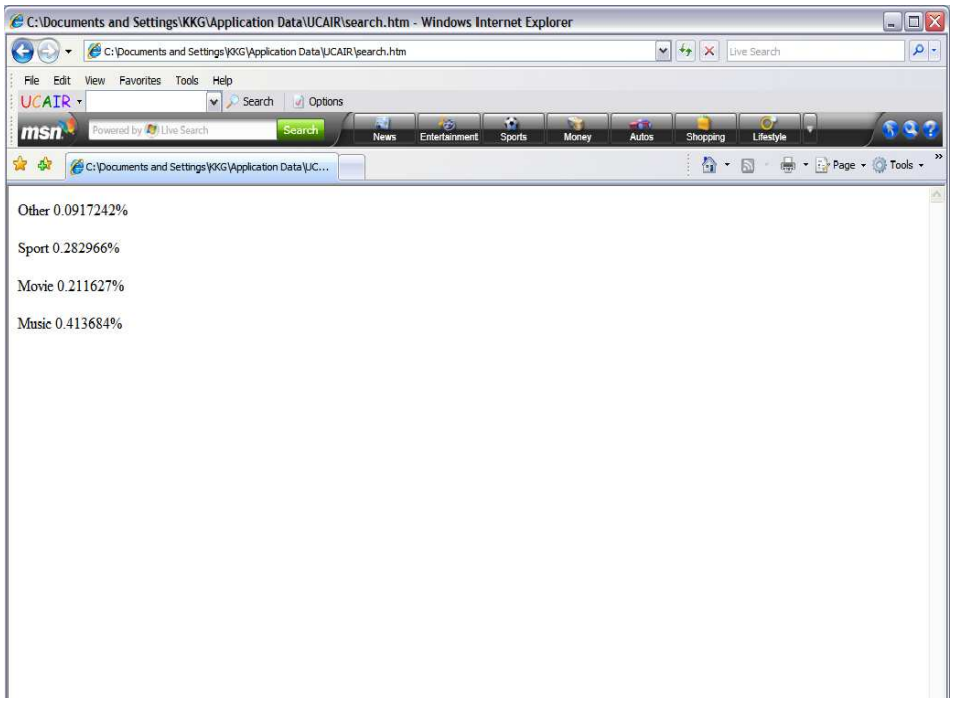

Figure 6.5: Result Page

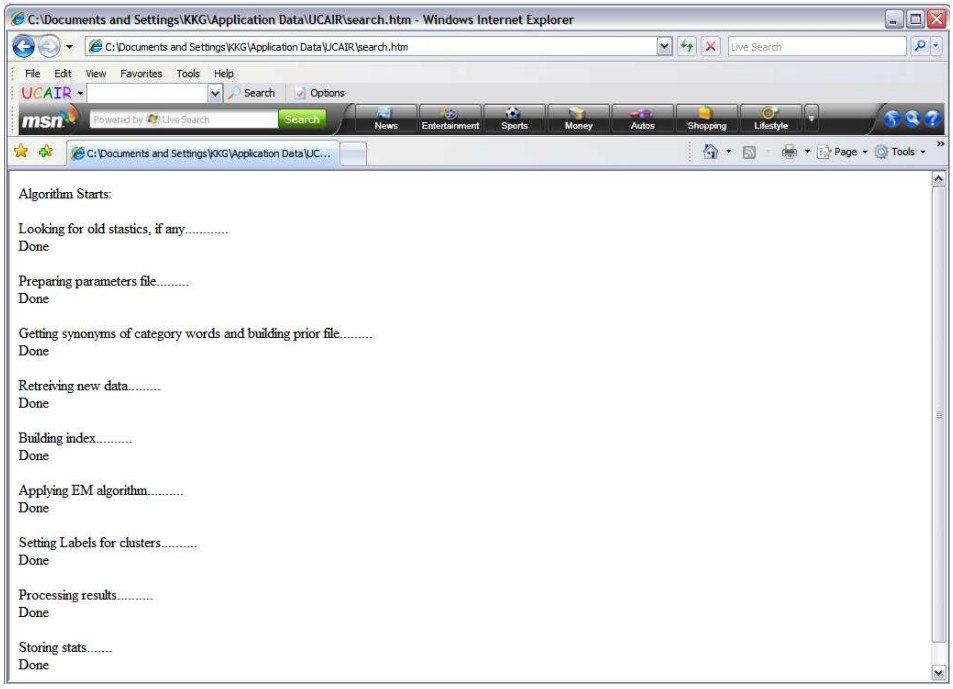

Figure 6.6: Run Page

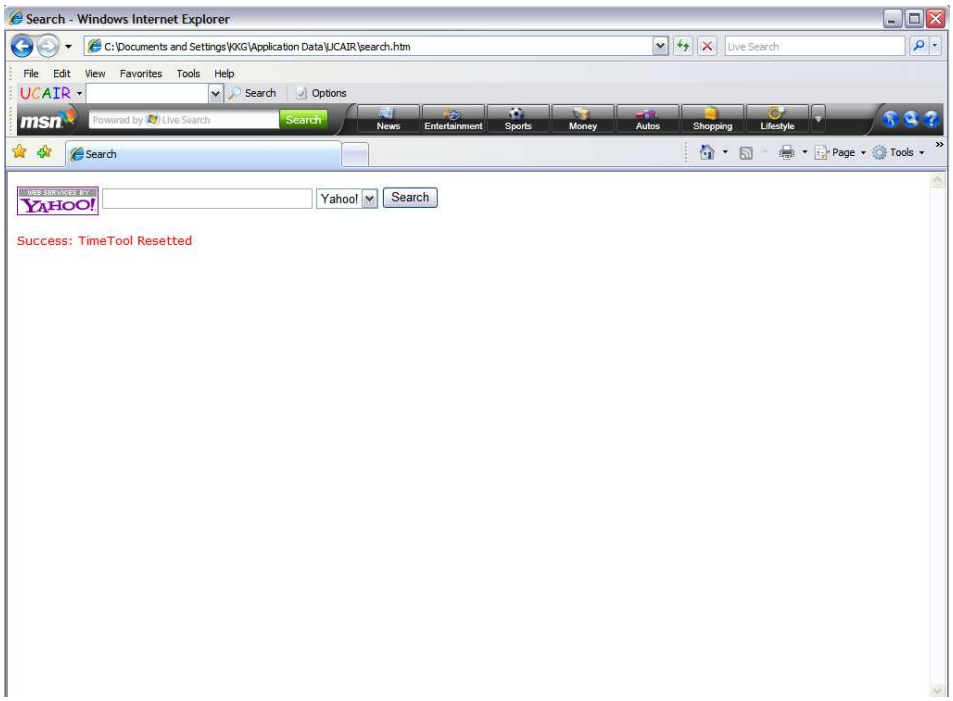

Figure 6.7: Reset Page

7 Evaluation of the System

In this section, we will talk about the Time tool results and discuss them in detail. To test our tool, we separately measure the performance of our tool in terms of detection and categorization.

In order to evaluate our tool, we perform a simulated study and randomly chose some users from AOL log and simulated their queries using UCAIR toolbar. We then used existing categorization results from Yahoo Directory to get labels of those URLs. Upon getting a subset of clicked documents for which we know labels, we ran our Time tool to evaluate the categorization.

In order to see the performance of the tool in subsequent runs, we divided the log in to two halves and ran twice for each user such that in the first run, only first part of the log is introduced. Also, we divided the log such that atleast one URL from each category appears in both halves unless we have only one URL in a particular category in which case, we placed it randomly.

In each run, we then finally labelled the category with Correct(C), Uncategorized(U), InCorrect(I) such that:

1. *Correct* - If the label matches the label of the Yahoo directory. If the url exist in multiple categories in Yahoo directory, we considered it as a correct if it was categorized in any one of the catgeory.

2. *Uncategorized* - If tool unable to clasify it into one of the specified categories and fail to detect any appropriate category.

3. *Incorrect* - If the label doesn't match with the Yahoo directory label. If the url exist in many categories in Yahoo directory, we considered it as incorrect if it doesn't fall into any of the multiple categories.

We also report the detected categories by the time tool. To evaluate the detection mechanism, we marked the detected categories as "Useful" or "Not Useful". Now, to mark them we again used the Yahoo directory and marked the detected category as useful if and only if there is atleast one URL in the log in that category.

In this study, we chose a total of 10 users to evaluate our tool. Table 7.1 shows the number of categories specified and number of categories detected

User	Categories specified	Categories detected
1	6	1
2	6	1
3	7	1
4	5	1
5	5	1
6	11	2
7	10	2
8	6	1
9	5	1
10	10	2

Table 7.1: Categories Statistics

User	Number of Urls in first run	Number of Urls in second run
1	21	8
2	6	17
3	26	10
4	4	10
5	5	1
6	21	10
7	7	27
8	9	5
9	7	22
10	12	60

Table 7.2: URL Statistics

by the tool for each user. In Table 7.2, we have shown the number of URLs i.e. size of the log in first and second run.

Table 7.3 and Table 7.4 show the categorization results of our study. It shows the Number of *correct*, *uncatgeorized*, *incorrect* labels for each user. We also plotted the corresponding graph shown in Figure 7.1. As we can see, the performance is much better in second run. This was quite expected since as our tool will process more and more log, it will enable the tool to build better and better prior, which will inturn assist Expectation-Maximization algorithm to perform better classification.

Finally, Table 7.5 shows the categories detected by our tool and their usefulness. As we can see, in most of the cases our tool has detected a *interesting* and an *useful* label and thus can assist users in managing their time.

User	Performance during first run
1	15 (Correct), 5 (Uncategorized), 1 (Wrong)
2	1 (Correct), 5 (Uncategorized), 0 (Wrong)
3	16 (Correct), 6 (Uncategorized), 4 (Wrong)
4	1 (Correct), 3 (Uncategorized), 0 (Wrong)
5	3 (Correct), 2 (Uncategorized), 0 (Wrong)
6	16 (Correct), 5 (Uncategorized), 0 (Wrong)
7	3 (Correct), 4 (Uncategorized), 2 (Wrong)
8	4 (Correct), 5 (Uncategorized), 0 (Wrong)
9	5 (Correct), 2 (Uncategorized), 0 (Wrong)
10	6 (Correct), 6 (Uncategorized), 0 (Wrong)

Table 7.3: Categorization results

User	Performance during second run
1	7 (Correct) , 1 (Uncategorized), 0 (Wrong)
2	10 (Correct) , 6 (Uncategorized), 1 (Wrong)
3	7 (Correct) , 3 (Uncategorized), 0 (Wrong)
4	4 (Correct) , 6 (Uncategorized), 0 (Wrong)
5	1 (Correct) , 0 (Uncategorized), 0 (Wrong)
6	8 (Correct) , 2 (Uncategorized), 0 (Wrong)
7	21 (Correct) , 4 (Uncategorized), 2 (Wrong)
8	2 (Correct) , 2 (Uncategorized), 1 (Wrong)
9	12 (Correct) , 10 (Uncategorized), 0 (Wrong)
10	43 (Correct) , 12 (Uncategorized), 5 (Wrong)

Table 7.4: Categorization results

User	Category detected	Useful (yes/no)
1	car	yes
2	fishing	yes
3	drugs	yes
4	bedding	yes
5	level	no
6	information, album	no, yes
7	car, furniture	yes, yes
8	music	yes
9	animal	yes
10	city, boat	yes, yes

Table 7.5: Detected categories and their usefulness

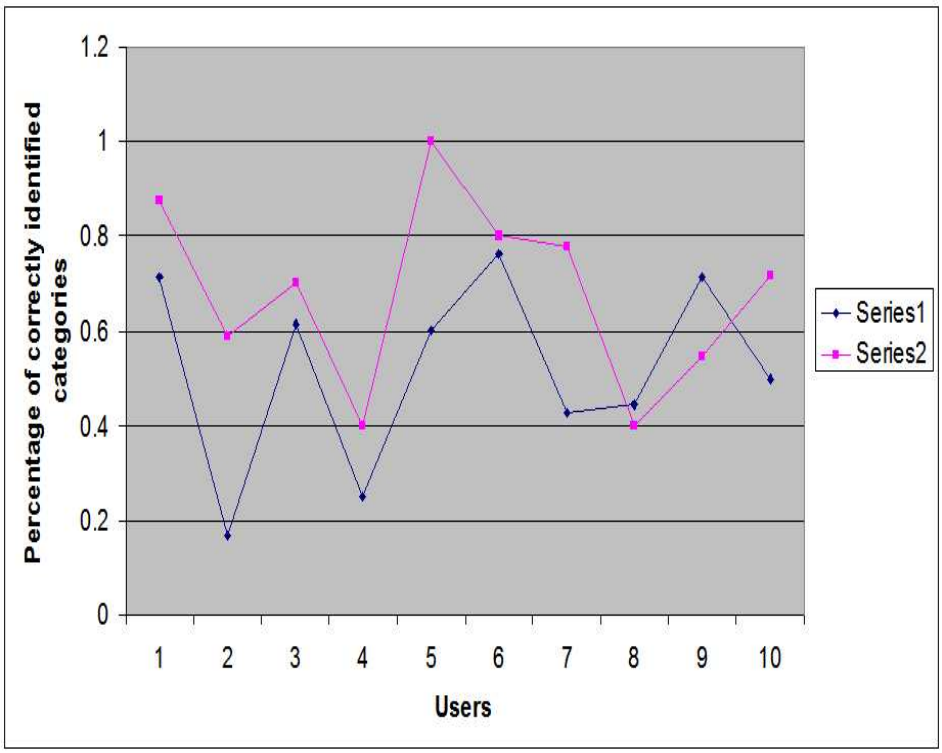

Figure 7.1: Performace variation in first and second run

8 Conclusions and Future Work

In this chapter, we conclude by summarizing our contributions, identifying some of the limitations of the present work and pointing out the directions for future work.

8.1 Contributions

We have developed an online algorithm that not only analyze the time spent by the user in pre-specified classes but also helps in detecting new categories. Our work takes advantage of the client side agent UCAIR user log to analyze the user activities.

The main problem we addressed is to track and detect data with no training data. In this study we studied various approaches. The approach taken by us is not limited to this application but can also be used in organizing stories, emails, library books and soon. Also, this is a very useful technique to recommend information. To the extent of our knowledge, this is first effort in this direction. Our tool is scalable, sustainable, automatic, quite generic, and can be easily integrated with any client side agent.

8.2 Limitations and Future Work

- One of the downside of our work is that we are only using search log to predict the time spent by the user in each category. However, in reality the user may not spend all of his time in front of his machine. Thus we need a mechanism that can keep track of User's physical activities as well.

  We already see a lot of handheld devices in the market with storage and GPS capability. If in future, these devices can track the current user activities and maintain a log of them and transfer them as user come close to his personal desktop/notebook, then these bigger machines, which usually have a vast amount of disk space, can manage, and store the user personal log. And hence both physical and virtual world

activities can then be combined and used with our tool to obtain more precise results.

- Our labeling of topic is only one level. However, it will be great if a tool can sub-categorise the categories itself in a tree like structure, for eg: If a user spends a total of 40% of his time in *Sports* activities and is interested in basketball, football and cricket, it is not clear though how much time user has spent in each of these games. However, if a tool can sub-categorize sports category into these three sub-categories, saying something like, (10% cricket, 10% football, 15% basketball and 5% other games), it will be more informative to the user.

- Right now, results of the time tool are not used back by the UCAIR toolbar to improve the search accuracy but we think that the results of the time tool can be very useful in not only recommending articles to the user but also in improving the search accuracy especially in the case of ambiguous queries. Consider an example where user types "Jaguar" and we know from time tool that user spends around 20% of his time in searching about animals and birds. Then we know that it is very likely that the user is searching for "Jaguar Cat" as opposed to the user who spends majority of his time in cars and automobiles, where he may be looking for luxury car manufacturer.

- The tool fails to classify those documents which doesn't have any text information apart from HTML tags, pictures, Javascript code etc.

References

[1] J. Allan, J. Carbonell, G. Doddington, J. Yamron, and Y. Yang. Topic detection and tracking pilot study: Final report. In *DARPA Broadcast News Transcription and Understanding*, 1998.

[2] D. Baker, T. Hofmann, A. McCallum, and Y. Yang. A hierarchical probabilistic model for novelty detection in text. *CMU*, 1999.

[3] World Wide Web Consortium. Xml. *http://en.wikipedia.org/wiki/XML*, 1998.

[4] Gabor Cselle, Keno Albrecht, and Roger Wattenhofer. Buzztrack: topic detection and tracking in email. In *IUI*, pages 190–197, 2007.

[5] J. Fiscus, G. Doddington, J. Garofolo, and A. Martin. Nist's 1998 topic detection and tracking evaluation. In *DARPA Broadcast News Workshop*, 1999.

[6] V. Lavrenko, J. Allan, E. DeGuzman, D. LaFlamme, V. Pollard, and S. Thomas. Relevance models for topic detection and tracking. In *HLT*, 2002.

[7] Juha Makkonen, Helena Ahonen-Myka, and Marko Salmenkivi. Applying semantic classes in event detection and tracking. In Rajeev Sangal and S. M. Bendre, editors, *Proceedings of International Conference on Natural Language Process ing (ICON 2002)*, pages 175–183, Mumbai, India, 2002.

[8] Juha Makkonen, Helena Ahonen-Myka, and Marko Salmenkivi. Topic detection and tracking with spatio-temporal evidence. In Fabrizio Sebastiani, editor, *Proceedings of 25th European Conference on Information Retrieval Research (ECIR 2003)*, pages 251–265. Springer-Verlag, 2003.

[9] Qiaozhu Mei, Xuehua Shen, and ChengXiang Zhai. Automatic labeling of multinomial topic models. In *SIGKDD*, pages 490–499, 2007.

[10] P. Mulbregt, I. van, L. Gillick, S. Lowe, and J. Yamron. Text segmentation and topic tracking on broadcast news via a hidden markov model approach. In *ICSLP*, volume 6, pages 2519–2522, 1998.

[11] K. Seymore and R. Rosenfeld. Large-scale topic detection and language model adaptation. *Carnegie Mellon University Technical Report*, 1997.

[12] Bin Tan. Ucair personalized search toolbar: Design and implementation. *MS Thesis*, 2005.

[13] Bin Tan, Xuehua Shen, and ChengXiang Zhai. Mining long-term search history to improve search accuracy. In *SIGKDD*, pages 718–723, 2006.

[14] Carnegie Mellon University and Amherst the University of Massachusetts. Lemur tookit. *http://www.lemurproject.org/*, 2008.

[15] Wikipedia. Strategy pattern. *http://en.wikipedia.org/wiki/Strategy_pattern*, 2008.

[16] Yiming Yang, Thomas Ault, Thomas Pierce, and Charles W. Lattimer. Improving text categorization methods for event tracking. In Nicholas J. Belkin, Peter Ingwersen, and Mun-Kew Leong, editors, *Proceedings of SIGIR-00, 23rd ACM International Conference on Research and Development in Information Retrieval*, pages 65–72, Athens, GR, 2000. ACM Press, New York, US.

[17] Yiming Yang, Jaime Carbonell, Ralf Brown, Tom Pierce, Brian T. Archibald, and Xin Liu. Learning approaches for detecting and tracking news events. *IEEE Intelligent Systems*, 14(4):32–43, 1999.