Context-aware Search for Personal Information Management Systems

Jidong Chen [1,2] Wentao Wu [1] Hang Guo [2] Wei Wang [1]

[1]Fudan University, Shanghai, China
[2]EMC Research China, Beijing, China

jidong.chen@emc.com, wentaowu@fudan.edu.cn,
hang.guo@emc.com, weiwang1@fudan.edu.cn

Abstract

With the fast growth of disk capacity in personal computers, keyword search over personal data (a.k.a. desktop search) is becoming increasingly important. Nonetheless, desktop search has been shown to be more challenging than traditional Web search. Modern commercial Web search engines heavily rely on structural information (i.e., hyperlinks between Web pages) to rank their search results. However, such information is not available in the circumstance of desktop search. Therefore, state-of-the-art desktop search systems such as Google Desktop Search usually leverage pure text-based ranking approaches (e.g., TF-IDF), which often fail to give promising rankings due to the misinterpretation of user intention.

We observed that in desktop search, the semantics of keyword queries are often context-aware, i.e., they are related to the current activity state (e.g., writing a paper, navigating a website, etc.) of the user. In this paper, we present a novel context-aware search framework by taking this activity information into consideration. Specifically, we use Hidden Markov Model (HMM) to capture the relationships between user's access actions (e.g., opening/closing files, sending/receiving emails, etc.) and activity states. The model is learned from user's past access history and is used to predict user's current activity upon the submission of some keyword query. We further propose a ranking scheme with this predicted context information incorporated. Experimental evaluation demonstrates both the effectiveness of the proposed context-aware search method and the enhancement to user's search experience.

1 Introduction

With the fast advance in storage technologies, disks with capacity from hundreds of gigabytes to even several terabytes are now very common in personal computers. As a result, the number of documents stored in the local file system also grows very quickly, and functionalities that support effective search for a particular document is of ever growing importance. However, achieving such kind of functionalities turns out to be quite challenging. Unlike in the case of Web search, where powerful ranking schemes such as PageRank [22] and HITS [19] can be employed, there is no structural information (i.e., hyperlinks between webpages) that can be directly used to rank documents inside personal computers in similar ways. Therefore, state-of-the-art desktop search engines, such as Google Desktop Search, usually adopt pure text-based ranking approaches (e.g., TF-IDF scores).

Nonetheless, TF-IDF scores do not always work well, for the following two reasons:

- *The same keyword query can have multiple meanings.* For example, "*apple*" may stand for the name of either a fruit or a company. "*Michael Jordan*" may refer to either a basketball player or a computer science professor. TF-IDF scores only consider the frequency that the keyword appears in the documents, and cannot distinguish such ambiguities. This problem has already been extensively studied in Web search. For instance, in [21], clickthrough data is analyzed to learn user's preference over different semantics of the query. But to the best of our knowledge, there is still no previous work on distinguishing query semantics in desktop search.

- *The goal of desktop search is to find some particular item.* Even if the keyword query is not ambiguous (i.e., its meaning is unique), a document with high TF-IDF score often does not mean it is the *particular* item that the user is trying to find. Consider, for example, a computer science student studying machine learning can keep hundreds of related papers in his personal computer. Suppose he wants to find some recent paper about Bayesian Network that he has downloaded from the ACM digital library. Using "*Bayesian Network*" as keywords may not help since there may be still dozens of papers talking about this topic, and those documents with high TF-IDF scores are most likely classic papers or surveys in this area. Note that this is quite different from Web search, in which users usually are interested in seeking *relevant pieces of information* (a.k.a. *navigation*). The goal of seeking particular items in desktop search has also been phrased as *re-finding known items* [12] in the literature.

We observed that, the semantics of keyword queries are often highly related to the *context* when they are issued. For example, a computer science student *John* may wish to search for a related paper written by "*Michael Jordan*" when he is writing his thesis, while he may be interested in the playing history of "*Michael Jordan*" when he is reading some NBA news. While intuitively this context information is valuable in addressing the first problem listed above, it is also helpful in resolving the second one. For instance, the paper about "*Bayesian Network*" may be downloaded several days ago when *John* was also writing his thesis. It thus provides important evidence for us to infer that this paper is highly relevant to the context "writing thesis". As a result, it is reasonable to boost the rank of this paper when the same context happens again.

A natural question is then whether it is practical to leverage this kind of context information. In the case of Web search, such context information usually cannot be automatically captured by Web search engines, since due to security consideration, Web browsers are not allowed to monitor user's behavior outside the browser. Therefore, the Web browser is unaware of context information such as "writing thesis", and this piece of information is lost when the query is issued. However, desktop search engines, on the other hand, do not have such security issues since everything they are concerned with is confined within user's personal computer. As a result, they can have more access privilege to the local system, and thus have higher chance to record this context information, which can then be probably used to help learn the semantics of the query.

In this paper, we address the problem of improving the quality of keyword search over personal data by taking context information into consideration. To achieve this, we have developed a novel desktop search system called *iMecho*. Figure 1 illustrates its architecture.

The *context* here is defined as the user's activity state, such as *writing a paper*, *navigating a website*, etc., which will be referred to as *task* in this paper. When performing a task, the user may have a series of access events. For example, when writing a paper, the user may first open the document file, then type in some text, and finally save and close the file. Before closing the file, he may also browse some web pages and copy a couple of references into the paper. Based on this observation, we propose to infer the current task by analyzing these access events. As shown in Figure 1, in *iMecho*, such events are recorded by various

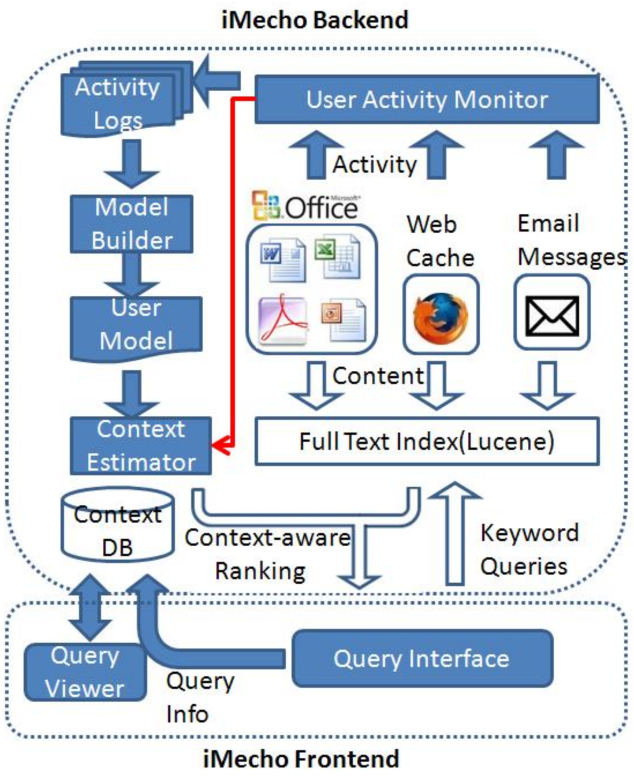


Figure 1: System Architecture

user activity monitors[1]. We then build a user model upon the events collected. As will be elaborated in Section 2, we use Hidden Markov Model (HMM) to capture the relationships between user's access events and (latent) tasks. The model is built and periodically updated offline. At runtime, when user submits a keyword query, his recent access events are used by the model to predict the current task. This information is finally incorporated into *iMecho*'s ranking scheme, which will promote those items that are more related to user's current task.

The rest of the paper is organized as follows: In Section 2 and Section 3, we discuss the user model and context-aware ranking scheme in details, respectively. Experimental results are presented in Section 4, and related work is summarized in Section 5. We conclude the paper in Section 6.

2 User Model

The user model lies in the core of *iMecho*. We assume that at any time, the user is performing *some* task, and we use an *empty* task to address the special case when the user is idle. Under this perspective, the behavior of the user could

[1]The user activity monitors are application-specific. For example, we use VSTO (Visual Studio Tools for Office) to implement monitors for Microsoft Word/Excel/Powerpoint/Outlook.

be viewed as a sequence of tasks. However, these tasks are *latent*. What could be directly observed and recorded are the access events of the user, which also consist of a sequence. It is then quite natural to make the Hidden Markov Model (HMM) [23] a good candidate for the user model, which elegantly captures the relationship between one observable sequence and one hidden sequence. Meanwhile, another basic hypothesis lying behind HMM is that the current task should only depend on its previous task, which also makes sense in most cases under the circumstances of desktop activities.

An HMM can be formally represented as $\lambda = (T, R, \pi, \mathcal{A}, \mathcal{B})$, in which

- T is a set of tasks;
- R is a set of *resources*, e.g., files, Web pages, emails, etc.;
- π stands for the initial probability vector of tasks, i.e., $\pi(i)$ is the probability that the user is currently performing task i when *no* access event is observed;
- $\mathcal{A}$ denotes the $|T| \times |T|$ transition matrix of tasks, i.e., $\mathcal{A}(i, j)$ is the transition probability from task i to task j;
- $\mathcal{B}$ is the $|T| \times |R|$ association matrix between tasks and resources, i.e., $\mathcal{B}(i, j)$ is the probability that resource j is accessed when the current task is i.

Training an HMM is a well-known process in research community. After appropriately initializing parameters $\pi, \mathcal{A}, \mathcal{B}$, running the Baum-Welch algorithm [25] will give us a locally optimized HMM $\lambda' = (T, R, \pi', \mathcal{A}', \mathcal{B}')$, with respect to the user activity log. This optimized HMM will be used as the user model in *iMecho*. According to previous researches (e.g., [23]), giving uniform initial estimates to the π and $\mathcal{A}$ parameters is adequate in almost all cases, which therefore is also done in *iMecho*. The challenge here is to give good estimates to the $\mathcal{B}$ parameter.

In this section, we elaborate how to estimate $\mathcal{B}$. Specifically, we propose a burst-based task mining framework to establish the relationships between tasks and resources. A *burst* is an impulse of access events in the user activity log, which intuitively indicates the start of a new task. The whole framework is divided into two steps: i) *burst detection*; and ii) *burst-task mapping*. Next, we first describe the user activity log, which is closely related to our framework, and then illustrate each of the above two steps in details.

2.1 User Activity Log Here we briefly describe the information we recorded in the user activity log L. Logically, L can be viewed as a sequence of access events. Each event e is represented as a triple $e = (r_e, t_e, a_e)$, where r_e is the *URI* (for *Uniform Resource Identifier*) of the resource, t_e is the time when the access event e occurs, and a_e is the particular type of *action*. For example, the action on a *Microsoft Word* document may be *Open*, *Close*, *Save*, etc.. For all the application monitors we implemented, r_e and t_e can be precisely recorded for every e. Actually, this can be done even if we do not develop specific monitors for most applications, since nearly all modern operating systems now provide some API to *notify* access events, which can be used to implement a monitor overseeing the entire file system. Due to this general setting, we assume that the availability of r_e and t_e is independent of our specific implementation in *iMecho*. However, a_e may be specific to different applications, depending on the API exposed by the software provider. Therefore, for applications that do not have fine-granularity event notification API, we simply set a_e to be the default type $Access$. The user model described in this paper will not rely on the availability of a_e, and we will discuss some possible refinements in Section 2.4.1 if a_e is also available.

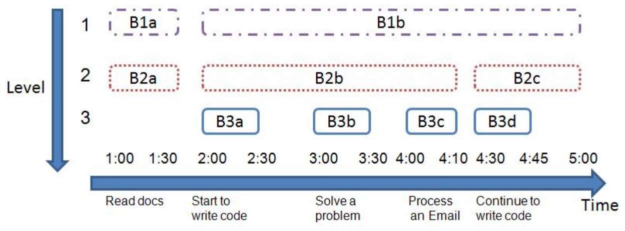


Figure 2: Bursts for John's one afternoon

2.2 Burst Detection We first use an example to illustrate the concept of a *burst*. Suppose Figure 2 shows the activities in one afternoon of *John*, our computer science student introduced in Section 1.

EXAMPLE 1. (JOHN'S ONE AFTERNOON) *At 1 pm John began to do some investigation on Machine Learning, he read some papers and web pages. That was the first burst of his activities. At around 1:30 he felt tired so he took a nap. There was no user activities when he was sleeping. Half an hour later he woke up and began to work on his demo program. He started his editor and opened several documents for references. At 2:30 he began to write the code. He focused on programming so he opened fewer documents at that time. At 3 he encountered some problem with a software package that was used in his code. Therefore he searched his PC and read some documents. Finally the problem was solved at 3:30. At 4 John received an important email from his co-author, who found a subtle issue in their paper. John had to pause his current task by saving the code already written. He read the paper mentioned in his co-author's email and got the problem. He remembered that the solution was in a web page. So he accessed Internet and found that web page. Then he replied to his co-author at 4:10. After that John made himself a cup of coffee and had some relax. Almost no activities were detected between 4:10* *and 4:30. John continued writing his code after 4:30, and there were quite a few activities from 4:30 to 4:45. Finally he finished his work at 5.*

In the above example, John conducted three tasks in that afternoon:

- t_1: *investigation on Machine Learning*, which was started at 1 and ended at 1:30.
- t_2: *code writing*, which was started at 2 and paused at 4. At 4:30 it was resumed and finally finished at 5.
- t_3: *replying email*, which was lasted from 4:00 to 4:10.

We observed that, each time a new task is started or resumed, there is a burst of activities. A burst can be part of a task, or a complete task. Previous research [18] has also shown that "the appearance of a topic in a document stream is signaled by a *burst of activity*, with certain features rising sharply in frequency as the topic emerges". Therefore, by employing some burst detection technology, it is possible to discover tasks from the stream of access events.

Formally, a *burst* b can be represented as a time interval $b = [\tau_s, \tau_e]$, where τ_s and τ_e are the start and end time of the burst, respectively. We use the burst detection algorithm proposed in [18]. Figure 2 also shows bursts detected by applying this algorithm[2]. Here, bursts could be located at different levels, with respect to their *intensiveness*. The intensiveness of a burst is measured by computing the length of the time gap between successive access events within the burst. The higher level the burst is at, the higher intensiveness it has. For example, the highest level (Level 3) in Figure 2 consists of time periods involving intensive activities of *John*, such as *writing code* and *replying email*.

Due to the nature of the algorithm, any burst at level $k + 1$ will be completely contained in some burst at level k. As a result, these bursts can be further organized into a hierarchical structure called *burst tree*, as shown in Figure 3. A burst b at level $k+1$ becomes a *child* of a burst b' at level k if b is contained in b'. The root of the tree is actually a *virtual* burst that spans the whole time interval.

Our next step is to identify tasks from the bursts detected.

2.3 Burst-Task Mapping As illustrated in Example 1, bursts at higher levels are usually signals of some tasks. However, the mapping from bursts to tasks is not one-to-one. Rather, it is quite common that a task may contain several bursts. For example, the *code writing* task (t_2) of *John* contains the bursts $B3a$, $B3b$ and $B2c$. On the other hand, it is very unlikely that a burst could belong to multiple tasks at the same time. Therefore, the mapping from bursts to tasks should be many-to-one.

[2]Each rectangle stands for a single burst. The label of the rectangle represents the identity and the level of the burst. For example, the first burst of level 1 is marked with $B1a$, the second burst of level 3 is marked with $B3b$, and so forth.

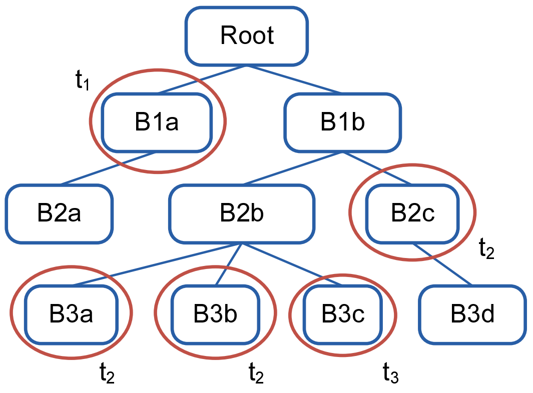


Figure 3: Burst Tree

Formally, let B and T be the set of bursts and tasks, respectively. Equivalently, B could also be viewed as the set of nodes in the burst tree. A *burst-task mapping* m is a *surjective* function from some subset $B' \subseteq B$ to T. A further comparison of Figure 2 and 3 leads to the following property of m.

PROPERTY 2.1. *Let* $Child(b) = \{c_1, c_2, ..., c_k\}$ *be the children of node* b *in the burst tree, and* $Child(b) \subseteq B'$. *If* $m(c_1) = m(c_2) = \cdots = m(c_k) = t \in T$, *then* $b \in B'$ *and* $m(b) = t$.

Intuitively, Property 2.1 says that, if all children of some burst b could be mapped to the same task t, then b itself could also be mapped to task t. This is due to the observation that, some *long* tasks may be paused and resumed several times during the whole time period.

EXAMPLE 2. (BURST-TASK MAPPING) *In Figure 3, suppose* $B2a$ *is mapped to* t_1; $B3a$, $B3b$, *and* $B3d$ *are mapped to* t_2; *and* $B3c$ *is mapped to* t_3. *Since* $B2a$ *is the only child of* $B1a$, *according to Property 2.1,* $B1a$ *could also be mapped to* t_1. *Similarly,* $B2c$ *could be mapped to* t_2, *since* $B3d$ *is its unique child.*

Property 2.1 then naturally suggests a bottom-up approach to find the burst-task mapping m. We first determine corresponding tasks for the bursts at the leaf nodes of the tree. We then climb up the tree and inductively determine the tasks for each non-leaf node, with respect to Property 2.1. The remaining problem is how to determine tasks for each leaf node. This is difficult, since at the beginning we even have no idea of how many tasks there are at all! The tasks t_1, t_2, and t_3 in Example 1 are actually the *result* from the task detection algorithm instead of the *prior* knowledge we have.

To overcome this issue, we need to use some information beyond merely *time* in the user activity log. Recall that as described in Section 2.1, the user activity log L also records the *URI* for each resource accessed in the event. Intuitively, if two bursts could be mapped to the same task, then it is highly likely that the sets of resources accessed in the two bursts are similar. Therefore we could identify tasks by checking the similarity of bursts.

Formally, let $R(b)$ be the set of resources accessed in the burst b[3]. Two bursts b_1 and b_2 are said to be *similar*, denoted as $Sim(b_1, b_2)$, if $s(R(b_1), R(b_2)) > \delta$, where $s(A, B)$ is some *similarity function* computing the similarity of the sets A and B, and δ is some prespecified threshold. Any reasonable similarity function could be used. In our implementation, we define

$$s(A, B) = \frac{|A \cap B|}{\max(|A|, |B|)}, \tag{2.1}$$

and set $\delta = 0.5$.

Algorithm 1 summarizes the task mining framework discussed so far. Line 3 to 5 first mark all $m(b)$ as Nil, i.e., the mapping from burst b to some task is not defined yet. Line 6 to 14 try to determine the mapping for leaf nodes of the burst tree, while Line 15 to 20 try to determine the mapping for non-leaf nodes, with respect to Property 2.1. When determining the mapping for a leaf node b, we check whether there is some other leaf burst b' that is similar to b and has already been mapped to some task t (line 8). If such a b' exists, we simply map b to t as well (line 9), otherwise we map b to a new task (line 11). Finally, we obtain tasks by merging corresponding bursts (line 21 to 30). A burst b is considered as (part of) some task if: i) $m(b)$ is determined (i.e., $m(b) \neq Nil$); and ii) $m(Parent(b))$ is not determined (i.e., $m(Parent(b)) = Nil$), where $Parent(b)$ is the parent burst of b in the burst tree. Figure 3 also shows the bursts that will be treated as (part of) some task in the end (circled by the red ellipses), after running Algorithm 1 on the burst tree. The sets of resources of these bursts will be merged to form the set of resources of the corresponding task (line 25). In Figure 3, $B1a$ will be merged to t_1; $B3a$, $B3b$ and $B2c$ will be merged to t_2; and $B3c$ will be merged to t_3.

The set of tasks T returned by Algorithm 1 is then used to initialize the HMM $\lambda = (T, R, \pi, \mathcal{A}, \mathcal{B})$, where $R = \cup_{t \in T} R(t)$. Suppose $t \in T$. For any resource $r \in R$, if $r \in R(t)$, we define $p(r|t) = n(r,t)/n(t)$, where $n(r,t)$[4] represents the number of accesses to r in task t, and $n(t) = \sum_{r' \in R(t)} n(r', t)$. If $r \notin R(t)$, we simply set $p(r|t) = 0$. This completes the initialization of the parameter $\mathcal{B}$, and we have discussed the initialization of the parameters π and $\mathcal{A}$ at the beginning of Section 2.

[3]We also use $R(t)$ to denote the set of resources accessed in the task t.

[4]$n(r,t)$ is obtained by summing up the number of events involving r in the corresponding bursts that are mapped to t.

Algorithm 1: Task Mining

```
   Input: $L$: the user activity log
   Output: $T$: the set of tasks
1  /* Let $\mathcal{T}$ be the burst tree. */
2  $\mathcal{T} \leftarrow BurstDetection(L)$
3  foreach $b \in Nodes(\mathcal{T})$ do
4      $m(b) \leftarrow Nil$
5  end
6  $i \leftarrow 1$
7  foreach $b \in Leafs(\mathcal{T})$ do
8      if $\exists b' \in Leafs(\mathcal{T})$ s.t. $Sim(b, b')$ and $m(b') = t$
       then
9          $m(b) \leftarrow t$
10     else
11         $m(b) \leftarrow t_i$
12         $i \leftarrow i + 1$
13     end
14 end
15 foreach $b \in NonLeafs(\mathcal{T})$ do
16     /* Let $Child(b) = \{c_1, c_2, ..., c_k\}$. */
17     if $m(c_1) = m(c_2) = \cdots = m(c_k) = t$ then
18         $m(b) \leftarrow t$
19     end
20 end
21 $T \leftarrow \emptyset$
22 foreach $b \in Nodes(\mathcal{T})$ do
23     if $m(b) \neq Nil$ and $m(Parent(b)) = Nil$ then
24         if $m(b) \in T$ then
25             $R(m(b)) \leftarrow R(m(b)) \cup R(b)$
26         else
27             $T \leftarrow T \cup \{CreateTask(b)\}$
28         end
29     end
30 end
31 return $T$
```

2.4 Discussion We discuss two related issues in this section, namely, i) possible refinements to tasks and ii) updates to the user model.

2.4.1 Task Refinements If more information is available in the user activity log, some possible refinements of the tasks could be performed. Here we only discuss the case when the action type a_e of an event e is known, since it is true in *iMecho* (although it is not required in the framework discussed in this section). Since we use some similarity function to determine whether we should merge two bursts according to the overlap of their sets of resources, it is likely that sometimes two bursts not merged could actually be mapped to the same task. The semantic information of action types may help in this situation. For example, if we find that resource r (e.g., some *Microsoft Word* document) is *opened* in burst b_1, but is never *closed* until burst b_2, where b_1 and b_2 are successive, then b_1 and b_2 are likely to be mapped to the same task, even if their similarity fails to pass the threshold δ.

2.4.2 Model Update The user model is trained using the user activity log. However, the access history will evolve over time, and as a result, new tasks beyond those used in the training stage will appear as time goes by. An immediate question is then how often the model should be updated. It is hard to imagine that a particular task would be repeated one year later. On the other hand, if we update the model every day, effective predication will then be difficult to achieve, since only limited number of repeated tasks could be observed. Therefore the time interval between two updates should be neither too long nor too short. We argue that one week may be a good choice and this setting is currently used in *iMecho*, although the user is allowed to change it freely. The reason is simply that people usually organize their work schedule week by week. A related question is whether there is certain number of repeated tasks during a week. Actually, according to the user study in [12], about 40% tasks are re-performed by the user within a week.

3 Context-aware Ranking

Based on the user model introduced in the previous section, we propose a context-aware ranking framework. Top-ranked items then stand for desktop resources that are most likely to be accessed next by the user at the time when he submits the query. In this section, we first formalize the context-aware ranking problem, and then propose our ranking function.

3.1 Problem Definition Suppose user submits a keyword query q at time k. Fix some length l[5] such that $0 \leq l \leq k$, and let L_k be the snippet of the user activity log consisting of events occurring in the time window $[k-l+1, k]$. The context-aware ranking problem is defined as follows:

DEFINITION 1. (CONTEXT-AWARE RANKING) *Given the user model $\lambda = (T, R, \pi, \mathcal{A}, \mathcal{B})$, the log snippet L_k, and the keyword query q, the context-aware ranking problem is to find some function $f : R \to \mathbf{R}^+ \cup \{0\}$ such that for any $r_1, r_2 \in R$, $f(r_1) > f(r_2)$ if and only if r_1 is more relevant than r_2, with respect to λ, q and L_k. Here $\mathbf{R}^+$ is the set of all positive real numbers.*

3.2 Ranking Function Without loss of generality, we could require that $0 \leq f \leq 1$. It is then quite natural to let f be the conditional probability $p(r|q, \lambda, L_k)$. We have

$$\begin{aligned} p(r|q,\lambda,L_k) &= \frac{p(r,q,\lambda,L_k)}{p(q,\lambda,L_k)} \\ &= \frac{\sum_{t\in T} p(r,q,t,\lambda,L_k)}{p(q,\lambda,L_k)}. \end{aligned} \tag{3.2}$$

Since $p(r,q,t,\lambda,L_k) = p(r,q|t,\lambda,L_k) \cdot p(t|\lambda,L_k) \cdot p(\lambda,L_k)$, and $p(q,\lambda,L_k) = p(q|\lambda,L_k) \cdot p(\lambda,L_k)$, we then have

$$p(r|q,\lambda,L_k) = \sum_{t\in T} \frac{p(r,q|t,\lambda,L_k)\cdot p(t|\lambda,L_k)}{p(q|\lambda,L_k)}. \tag{3.3}$$

What's more, since

$$\begin{aligned} p(r,q|t,\lambda,L_k) &= \frac{p(r,q,t,\lambda,L_k)}{p(t,\lambda,L_k)} \\ &= \frac{p(q,t|\lambda,L_k,r)\cdot p(\lambda,L_k,r)}{p(t,\lambda,L_k)}, \end{aligned} \tag{3.4}$$

and we assume that given λ, L_k and r, q and t are independent of each other, therefore

$$p(q,t|\lambda,L_k,r) = p(q|\lambda,L_k,r)\cdot p(t|\lambda,L_k,r). \tag{3.5}$$

Substituting Eq. (3.4) and (3.5) into Eq. (3.3), and notice the fact that

$$p(r|t,\lambda,L_k) = \frac{p(t|\lambda,L_k,r)\cdot p(\lambda,L_k,r)}{p(t,\lambda,L_k)},$$

we obtain

$$p(r|q,\lambda,L_k) = \sum_{t\in T} \frac{p(q|\lambda,L_k,r)\cdot p(r|t,\lambda,L_k)\cdot p(t|\lambda,L_k)}{p(q|\lambda,L_k)}. \tag{3.6}$$

Clearly, q is independent of λ and L_k. Thus

$$p(q|\lambda,L_k) = p(q). \tag{3.7}$$

On the other hand,

$$p(q|\lambda,L_k,r) = \frac{p(q,\lambda,L_k,r)}{p(\lambda,L_k,r)} = \frac{p(q,\lambda,L_k|r)}{p(\lambda,L_k|r)}, \tag{3.8}$$

and again due to the independence of q with λ and L_k,

$$p(q,\lambda,L_k|r) = p(q|r)\cdot p(\lambda,L_k|r). \tag{3.9}$$

Therefore, by substituting Eq. (3.7), (3.8), and (3.9) into Eq. (3.6), we get

$$p(r|q,\lambda,L_k) = \frac{p(q|r)}{p(q)} \cdot \sum_{t\in T}[p(r|t,\lambda,L_k)\cdot p(t|\lambda,L_k)]. \tag{3.10}$$

[5]We set $l = 10$ in *iMecho* by default, which is adjustable.

Since $p(q|r) \cdot p(r) = p(r|q) \cdot p(q)$, we finally have

(3.11)
$$p(r|q, \lambda, L_k) = \frac{p(r|q)}{p(r)} \cdot \sum_{t \in T} [p(r|t, \lambda, L_k) \cdot p(t|\lambda, L_k)].$$

Eq. (3.11) can be interpreted quite naturally. $p(r|q)$ could be viewed as the relevance of r and q when context information is not considered, while $p(r)$ is the prior probability that r is accessed even without the query q. The summation in Eq. (3.11) is the relevance of r and the current context information of the user, namely, the probability that user may next access r with respect to the user model λ and the most recent access history L_k.

For notational convenience, we now define

(3.12)
$$f_1(r) = \frac{p(r|q)}{p(r)},$$

and

(3.13)
$$f_2(r) = \sum_{t \in T} [p(r|t, \lambda, L_k) \cdot p(t|\lambda, L_k)].$$

As a result, $f(r) = p(r|q, \lambda, L_k)$ could be simply rewritten as

(3.14)
$$f(r) = f_1(r) \cdot f_2(r).$$

However, directly using Eq. (3.14) suffers some problems. First, $f_1(r)$ and $f_2(r)$ may have different magnitude scales. Second, user may desire some control over $f(r)$, by making the *context-free* part (i.e., $f_1(r)$) and the *context-aware* part (i.e., $f_2(r)$) tunable. We thus introduce some *tuning factor* α ($0 \leq \alpha \leq 1$), and refine $f(r)$ to be

(3.15)
$$f(r) = f_1(r)^{1-\alpha} \cdot f_2(r)^{\alpha}.$$

In our implementation, given $\lambda = (T, R, \pi, \mathcal{A}, \mathcal{B})$, L_k, and q, to compute $f_1(r)$, we set $p(r|q) = s(r, q)$, where $s(r, q)$ represents the cosine similarity score (i.e., TF-IDF score) between r and q, which is widely used in term-vector based information retrieval models. $s(r, q)$ could be directly obtained by querying the full-text index in *iMecho*, which is implemented by using the Lucene[6] library (See Figure 1). $p(r)$ is simply set to be $\frac{1}{|R|}$, i.e., we do not have prior knowledge of user's preference over resources, which is hard to measure. In practice, we could even set $p(r) = 1$ since it does not affect the ranking result, and we then simply have $f_1(r) = s(r, q)$. To compute $f_2(r)$, we first get the current task t_{now} by applying the well-known Viterbi algorithm [23] to λ and L_k. We then compute each summand in $f_2(r)$ by setting $p(t|\lambda, L_k) = p(t|t_{now})$, which could be obtained from $\mathcal{A}$, and setting $p(r|t, \lambda, L_k) = p(r|t)$, which could be obtained from $\mathcal{B}$.

[6]http://lucene.apache.org/java/docs/index.html

4 Experimental Evaluations

We now experimentally evaluate our context-aware search approach from two aspects: i) the search overhead, and ii) the effectiveness of the context-aware ranking.

4.1 Experiment Design

As pointed out in [12], evaluation of personal search systems is an extremely difficult task, due to the privacy issues concerning personal information. Meanwhile, incorporating tasks into the evaluation makes the problem even harder, since different people may understand the concept of task in slightly different ways. For instance, in Example 1, some people may not treat t_3 as a task.

To overcome the privacy issue, we developed a *logging* module (different from the application monitors and user activity log) in *iMecho* to record the query history of the user, which could be browsed by the user in the *query viewer* (see Figure 1). To overcome the multiple interpretation problem of tasks, we have to do some individual user studies.

We invited 10 volunteers to take part in the experiment. Each of the participants installs *iMecho* on his own personal computer. We first help them be familiar with *iMecho*'s functionality and user interface. Each user then uses *iMecho*'s indexing module to index the resources (files, emails, etc.) that are expected to be related to the user's daily work in the following week. *iMecho* will also automatically update the index when new resources appear in the directories specified by the user. The experiment lasts for one week. The user activity log of the first three days is used to train the model, and the system is tested in the later two days.

During the training phase, the participants just perform their daily work as usual. In the testing phase, participants are encouraged to submit 10 queries to *iMecho* when they want to search some resource in their computer. For each query submitted, *iMecho*'s logging module will record the information of the query, the set of resources retrieved and their rankings, and also user's click actions on these resources. To see the effect of the tuning factor α in the ranking function, we have to ask participants to adjust α to be 0, 0.2, 0.5, and 0.8, respectively, and check ranking results for each of these rankings. By setting $\alpha = 0$, we actually ignore the *context-aware* part in the ranking function, and in this case the items are purely ranked by their TF-IDF scores (i.e., $f(r) = f_1(r) = s(r, q)$). After some anonymization (e.g., replacing *URI*'s by integers), such information is collected to analyze the effectiveness of the proposed ranking framework.

4.2 Performance Evaluation

Due to the diversity of the configuration of participants' personal computers and the privacy issue, here we only report some performance statistics on a typical laptop with 2.0 GHz Intel Dual Core CPU and 2GB main memory, which is used by one of the authors (not in the 10 participants).

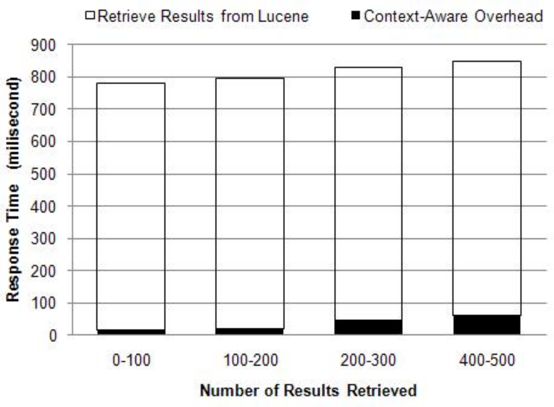


Figure 4: Comparison of the Average Overhead

Data Set. *The data set contains 9,431 desktop files in 1,019 directories. The average directory depth is 9 with the longest being 15. On average, directories contain 10.3 subdirectories and files, with the largest containing 241 ones. 75% of the files are smaller than 16 KB, and 95% of the files are smaller than 40 KB. The largest file is of size 21.5MB. The user log produced by the event monitor records totally 1,601 desktop events.*

Since the user's context is computed at the query time, there is some overhead for query context estimation. We compare the overhead with the search cost of Lucene. The results are given in Figure 4. In the case of desktop search, it is not common that the number of search results will grow to above 500. Therefore the overhead introduced by the context-aware search module is trivial (less than 10%) comparing to overall search costs.

4.3 Effectiveness Evaluation We next compare the effectiveness of our context-aware ranking scheme with the ranking scheme based on TF-IDF score (i.e., ranking scheme used by Lucene and other commercial desktop systems). Our evaluation consists of two parts. First, we present a case study of some user's search experience with *iMecho*, and the information released here is upon his agreement. Second, we analyze the feedbacks (i.e., clickthrough history recorded by *iMecho*'s logging module) on the search results from all the 10 participants, to give quantitative comparisons between the two ranking schemes.

4.3.1 Case Study Participant A issued the query "*PageRank model*" twice in the testing phase of the experiment when he was performing two different tasks:

- Task 1: To implement the PageRank model in his program.
- Task 2: To compare different ranking models.

Though the query terms are the same, they were used for quite different purposes. His first task was to write some code that implements the PageRank algorithm, while the second task was to compare PageRank with other ranking models like HITS.

The ranking results of the two queries are given in Table 1. To save space, only the top 5 results are shown in each case. The first row gives the results ranked by pure TF-IDF scores got from Lucene (i.e., by setting $\alpha = 0$), while iMecho's results (by setting $\alpha = 0.8$) under Task 1 and 2 are shown in the second and third row, respectively. Both the final score (i.e., $f(r)$) and the context-aware part of the score (i.e., $f_2(r)$) are shown in the last column of Table 1.

Here are some explanations about the results. The ranking in the second row (when the user was in Task 1) is exactly the same as the ranking returned from Lucene. This is because no results were ever accessed by the user when the query was issued, which means these resources are not closely relevant to Task 1. But when the user was in Task 2, according to Table 1, the context-aware scores ($f_2(r)$) of some documents are increased although their context-free scores ($f_1(r)$) are relatively low. User A recalled that he accessed these promoted documents when he was in Task 2.

The above case study shows that taking context information into consideration could sometimes better serve the user queries by correctly predicting user's search intention. Traditional desktop search engines will return the same results under both Task 1 and Task 2, which are not what the user wants in both cases.

4.3.2 Quantitative Comparison We next quantitatively compare the effectiveness of the two ranking schemes, namely, the ranking scheme based on pure TF-IDF scores and the one by also considering context information.

Precision and Recall

We first apply the well-known measures *precision* and *recall* to evaluate ranking quality. In general, precision measures the ability of a system to return *only* relevant results. It is defined as:

$$\text{Precision} = \frac{\text{\# of \textbf{relevant} results}}{\text{\# of results returned by the scheme}}.$$

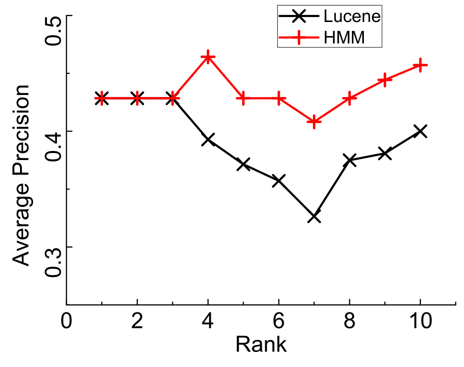
On the other hand, recall measures the ability of a system to return *all* relevant results, and is defined as:

$$\text{Recall} = \frac{\text{\# of \textbf{relevant} results}}{\text{\# of \textbf{relevant} results returned by \textbf{both} schemes}}.$$

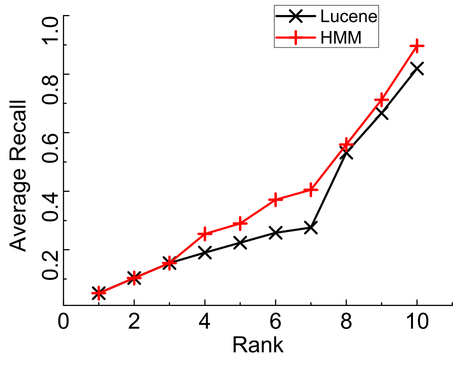
The definition of recall here follows [9], since in general it is extremely difficult to measure the total number of

Table 1: The ranking results of User A

Ranking Scheme	Ranking Result List	$f_1(r)$	$f(r)$ / $f_2(r)$
Lucene (TF-IDF) i.e. $\alpha = 0$	Jena-2.5.5/doc/javadoc/com/hp/hpl/jena/rdf/model/Model.html	0.896	N/A
	openrdf-sesame-2.1.3/docs/system/ch03.html	0.786	
	openrdf-sesame-2.1.3/docs/system/ch06.html	0.709	
	Jena-2.5.5/doc/images/Ont-model-layers-import.png	0.616	
	Jena-2.5.5/doc/images/Ont-model-layers.png	0.616	
iMecho (T1) $\alpha = 0.8$	Jena-2.5.5/doc/javadoc/com/hp/hpl/jena/rdf/model/Model.html	0.896	0.0246 / 0.010
	openrdf-sesame-2.1.3/docs/system/ch03.html	0.786	0.0239 / 0.010
	openrdf-sesame-2.1.3/docs/system/ch06.html	0.709	0.0234 / 0.010
	Jena-2.5.5/doc/images/Ont-model-layers-import.png	0.616	0.0227 / 0.010
	Jena-2.5.5/doc/images/Ont-model-layers.png	0.616	0.0227 / 0.010
iMecho (T2) $\alpha = 0.8$	The PageRank Citation Ranking- Bringing Order to the Web (1998).pdf	0.142	0.289 / 0.345
	Inside PageRank.pdf	0.427	0.263 / 0.233
	RandomWalks.ppt	0.267	0.211 / 0.199
	Jena-2.5.5/doc/javadoc/com/hp/hpl/jena/rdf/model/Model.html	0.896	0.0019 / 0.0004
	openrdf-sesame-2.1.3/docs/system/ch03.html	0.786	0.0018 / 0.0004



(a) Average precision



(b) Average recall

Figure 5: Comparison of average precision and recall

relevant resources to the query in the entire system. Instead, we define this set of relevant resources to be the *union* of the results treated as *relevant* by the user under each ranking scheme.

For each query, we get the top 10 results returned by each ranking scheme. The results that are actually clicked by the users are marked as *relevant*. We then compute average precision and recall over the 100 queries in total for both ranking schemes.

Figure 5 compares the average precision and recall of top-k results returned by both ranking schemes, where k increases from 1 to 10. For the HMM based context-aware ranking scheme used in *iMecho*, the statistics are computed over the results returned when setting $\alpha = 0.5$. We can see that the context-aware ranking method outperforms the TF-IDF based ranking method in terms of both precision and recall.

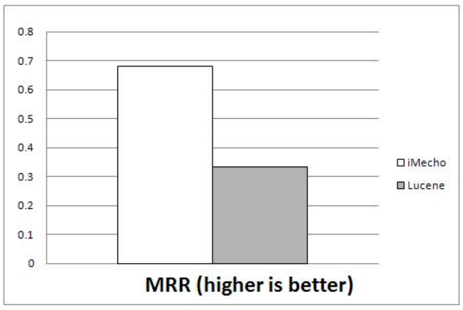
MRR and Top-k% Clicks

We then use the *Mean Reciprocal Rank* [7] (MRR for short) and the *Top-k% Clicks* as the metrics. In our scenario, the *reciprocal rank* of a query is defined as the reciprocal of the rank of the *first* relevant result. The mean reciprocal rank is then the average of the reciprocal ranks over all the queries. Formally, supposing there are N queries $q_1, \ldots, q_N$ and the rank of user's first choice in the result list of q_i is r_i, MRR is computed as:

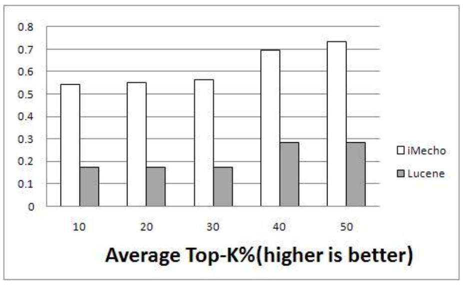
$$\text{MRR} = \frac{1}{N} * \sum_{i=1}^{N} \frac{1}{r_i}.$$

The *top-k% clicks* metric denotes the percentage of the user's clicks that fall into the top k% results returned by the ranking algorithm. For example, if the user chooses the first and third result out of the ten retrieved results, the top-10% and top-20% clicks are 0.5, while the top-30% clicks (and

[7] http://en.wikipedia.org/wiki/Mean_reciprocal_rank

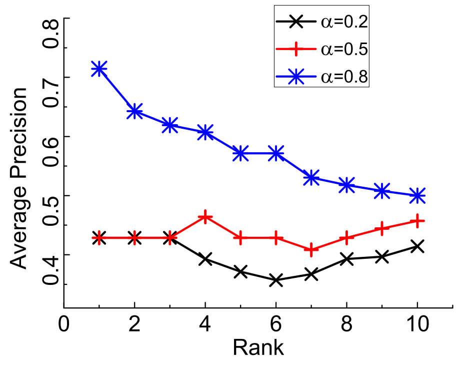


(a) MRR

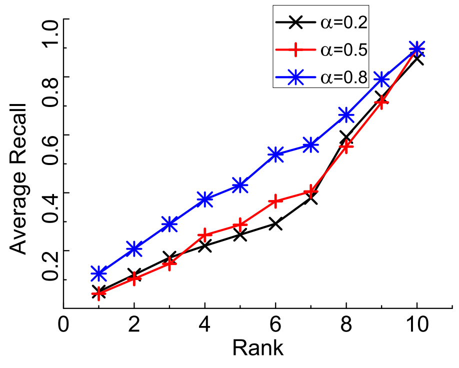


(b) TOPK

Figure 6: MRR and Top $k\%$ Clicks



(a) Average precision



(b) Average relative recall

Figure 7: Average precision and recall with different α's

also top-$k\%$ clicks with $k > 30$) is 1. Formally, we define

$$\text{Top-k\% Clicks} = \frac{\text{\# of clicks in top k\% of the results}}{\text{\# of total clicks}}.$$

Intuitively, MRR and top-$k\%$ clicks measure the user's preference over the top-ranked results. Figure 6(a) shows the MRR of *iMecho* ($\alpha = 0.5$) and Lucene, respectively. It is clear that iMecho's ranking results are more favorable by the users. The average top-$k\%$ clicks of iMecho and Lucene are shown in Figure 6(b). We can see that *iMecho*'s top-10% clicks is almost 2 times higher than that of Lucene. We thus conclude that top ranked results in *iMecho* are more preferable than those in Lucene.

Impact of the tuning factor α

We further investigate the impact of the tuning factor α to the ranking results in *iMecho*. As shown in Figure 7, when α increases, i.e., the weight of context-aware part of the score is increased, the precision is significantly improved while the recall is also slightly improved as well. This implies that many times, our ranking scheme is successful in predicting user's search intention when the query is issued.

5 Related Work

In our previous work [8] and [6], we developed XSearcher (renamed as the current name *iMecho* in [6]), an associative memory based desktop search system to enhance traditional keyword-based desktop search. However, the goal of XSearcher is quite different from the work described in this paper. XSearcher focuses on helping users in finding resources accessed a *long* time ago. The basic idea is to establish semantic associations among resources. For example, a document can be associated with an email if it once served as an attachment of that email. The key insight here is that, the associations among resources can play the same role as hyperlinks among Web pages, and ranking algorithms akin to PageRank could then be applied in desktop search. In this way, it is possible for users to find relevant items that may *not* contain the keywords in the issued query, as long as they are reachable by following paths starting from some resources hit by the keywords. Nonetheless, the approach used in that work cannot solve the two problems highlighted in Section 1, and the work in this paper can be viewed as complementary to that work, which helps users better seeking resources *recently* accessed by considering the context information. Figure 8 illustrates the big picture of our work on personal search. The effectiveness of the iMecho system by leveraging the context-aware search framework introduced in this paper has also been shown in our very recent demo [7].

Many research prototypes and commercial systems have been developed to support various forms of search on per-

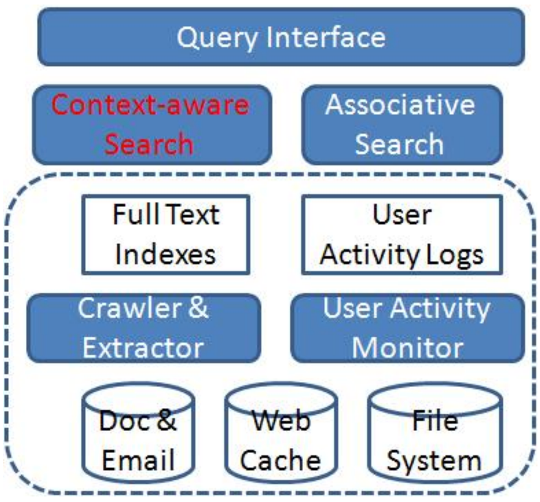


Figure 8: The big picture of our work on personal search

sonal desktop resources. The most prominent desktop search applications for Windows include Google desktop search and Microsoft Windows Desktop Search, and the Beagle open source project for Linux [1]. Apple Inc. also integrated an advanced desktop search application (named *Spotlight Search*) into their operating system, *Mac OS Tiger*. All these industrial desktop search engines support search over a variety of file types, yet none of them incorporates personalization, i.e. no user preferences are discovered to provide personalized results. In addition, there are currently only limited insights into the question of how to rank desktop search results. When ranking is available in some of these applications, it is usually performed according to some variations of TF-IDF, a textual-relevance-based criterion used in classic information retrieval.

Some PIM systems have been constructed in order to facilitate re-finding of various stored resources on the desktop. Stuff I've Seen [11] for example provides a unified index of the data that a person has seen on his computer, regardless of its type. Based on the fact that the user has already seen the information, contextual cues such as time, author, thumbnails and previews can be used to search for and present information. Similarly, MyLifeBits [13] targets storing locally all digital media of each person, including documents, images, audio and videos. They organize these data into collections and connect related resources with links. Haystack [16] emphasizes the relationship between a particular individual and her corpus. It automatically creates connections between documents with similar content and it exploits usage analysis to extend the desktop search results set. Feldspar [5] is a link-based desktop search prototype with new interface. Users can propose associative queries via a well-designed interface rather than simple keywords. But user activities are not tracked in Feldspar for associations and no activity-based associations are mined from user access patterns. Semex [10] is also a link-based personal information system. It employs a fancy reference reconciliation algorithm to integrate data from different sources and construct content-based associations by extracting metadata. Beagle++ [9] is a semantic desktop search prototype, which proposed various activity specific heuristics to generate links between resources that associate desktop resources. Beagle++ also logs user activities, such as attachment saving and file downloading, to generate associations, but it only focuses on associations from predefined user actions between web pages, email messages and files, not mining from a sequence of user access activities. Their approach was also limited to specific desktop contexts (e.g., publications, or web pages), whereas in our methods we explore much more general information sources such as file access patterns, which are applicable to any desktop resource.

Unlike traditional "one-size-fits-all" search engines, personalized search systems attempt to take into account preferences of individual users in order to improve the relevance of search results and the overall retrieval experience. Personalization techniques have been developed in diversified ways for web search. In a nutshell, the techniques can be classified into three categories, namely, *content based personalization*, *link-based personalization*, and *function-based personalization* [17]. Content-based personalization deals with the "relevance" measure of Web pages and the user's queries. In this approach, the query is modified to adapt the search results for the specific user. In order to manage user interests, a content-based personalization technique is used to construct user profiles, which store user interests derived from each user's search history [20]. Link-based personalization performs personalization based on link analysis techniques. Traditional link analysis techniques, like the PageRank algorithm, compute scores that reflect a "democratic" importance with no preferences in regard to any particular pages. However, in reality, a user may have a set of preferred pages in mind. The link-based personalized searching techniques redefine the importance of Web pages according to different users' preferences such as bookmarks as a set of preferred pages [14]. The function-based personalization first discovers user preferences on the search results from clickthrough data and then the ranking function is optimized according to the discovered preferences [15, 21]. However, these personalized techniques still miss contextual information often resulting or inferable from explicit and implicit user activities. For the personalization in desktop search, the desktop environment is comparably "limited" in the sense that we will be able to describe most relevant contexts more easily. It is possible to obtain the complete trace of user activity on his desktop, and therefore his accurate interests, goals, and preferences can be discovered for the context-aware search.

Context-aware search has been mainly presented in web search for query suggestions [4] and query classification [2]. Search contexts from users' search or browsing logs are effective for disambiguating Web queries and can help improve the quality of multiple search services [20]. However, in their approaches, only clickthrough information is considered as an important part of context. They also take little consideration of the user behavior at the time of querying. Though a recent work [3] also applied the HMM model in context-aware search, it only focused on Web search by considering only the query strings and simple clickthrough data. In comparison, we exploit the task context by analyzing the access events in the user activity log and use the Hidden Markov Model to capture the relationships between tasks and resources, which models the user's behavior at a higher level of semantics. In [24], besides clickthrough information, the authors also propose to treat preceding queries as implicit user feedback when considering the ranking of the documents. Nonetheless, the study was still confined in the sense of Web search, and it could be an interesting future work for *iMecho* to also incorporate preceding user queries as additional context information.

6 Conclusion

In this paper, we propose an HMM based context-aware search approach to help users improve search experience on personal desktop systems. The user model is built by analyzing the access events recorded in the user activity log, which captures the semantic relationship between user behaviors (*task*s in this paper) and resources. In particular, we propose a task mining algorithm to detect tasks from the log, which are then used to initialize the user model. Based on this model, we further propose a novel ranking scheme by taking the context information into consideration when answering user queries. The proposed framework has been implemented in the *iMecho* system, and the experimental evaluation shows that many times *iMecho* could outperform traditional TF-IDF based ranking schemes.

References

[1] Gnome beagle desktop search. http://www.gnome.org/projects/beagle/.

[2] H. Cao, H. Hu, D. Shen, D. Jiang, J. Sun, E. Chen, and Q. Yang. Context-aware query classification. In *SIGIR*, 2009.

[3] H. Cao, D. Jiang, J. Pei, E. Chen, and H. Li. Towards context-aware search by learning a very large variable length hidden markov model from search logs. In *WWW*, 2009.

[4] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *KDD*, 2008.

[5] D. Chau, B. Myers, and A. Faulring. What do do when search fails:finding information by association. In *CHI*, 2008.

[6] J. Chen, H. Guo, W. Wu, and W. Wang. imecho: an associative memory based desktop search system. In *CIKM*, pages 731–740, 2009.

[7] J. Chen, H. Guo, W. Wu, and W. Wang. imecho: a context-aware desktop search system. In *SIGIR*, pages 1269–1270, 2011.

[8] J. Chen, H. Guo, W. Wu, and C. Xie. Search your memory ! - an associative memory based desktop search system. In *ACM SIGMOD*, 2009.

[9] P. Chirita, S. Ghita, W. Nejdl, and R. Paiu. Beagle++: Semantically enhanced searching and ranking on the desktop. In *ESWC*, 2006.

[10] X. Dong and A. Halevy. A platform for personal information management and integration. In *CIDR*, 2005.

[11] S. Dumais, E. Cutrell, J. Cadiz, G. Jancke, R. Sarin, and D. Robbins. Stuff i' ve seen: a system for personal information retrieval and re-use. In *SIGIR*, 2003.

[12] D. Elsweiler and I. Ruthven. Towards task-based personal information management evaluations. In *Proceedings of the international ACM SIGIR conference on research and development in information retrieval*, pages 23–30, 2007.

[13] J. Gemmell, G. Bell, R. Lueder, S. Drucker, and C. Wong. Mylifebits: fulfilling the memex vision. In *Proceedings of the ACM Conference on Multimedia*, 2002.

[14] T. Haveliwala. Topic-sensitive pagerank. In *WWW*, 2002.

[15] T. Joachims. Optimizing search engines using clickthrough data. In *ACM SIGKDD*, 2002.

[16] D. Karger. Haystack: A customizable general-purpose information management tool for end users of semistructured data. In *CIDR*, 2005.

[17] Y. Ke, L. Deng, W. Ng, and D. L. Lee. Web dynamics and their ramifications for the development of web search engines. *Comput. Netw. J. (Special Issue on Web Dynamics)*, 50:1430–1447, 2005.

[18] J. Kleinberg. Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery*, 7(4):373–397, Octomber 2003.

[19] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.

[20] F. Liu, C. Yu, and W. Meng. Personalized web search for improving retrieval effectiveness. *IEEE Trans. Knowl. Data Eng.*, 16:28–40, 2004.

[21] W. NG, L. Deng, and D. Lee. Mining user preference using spy voting for search engine personalization. *ACM Transactions on Internet Technology*, 7(4), 2007.

[22] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bring order to the web. Technical report, Stanford University, 1998.

[23] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[24] X. Shen, B. Tan, and C. Zhai. Context-sensitive information retrieval using implicit feedback. In *SIGIR*, pages 43–50, 2005.

[25] L. Welch. Hidden markov models and the baum-welch algorithm. *IEEE Information Theory Society Newsletter*, 53(4):1–13, 2003.