Mining User Preference Using Spy Voting for Search Engine Personalization

Wilfred Ng, Lin Deng and Dik Lun Lee
Department of Computer Science and Engineering
Hong Kong University of Science and Technology

This paper addresses search engine personalization. We present a new approach to mining a user's preferences on the search results from clickthrough data and using the discovered preferences to adapt the search engine's ranking function for improving search quality. We develop a new preference mining technique called SpyNB, which is based on the practical assumption that the search results clicked on by the user reflect the user's preferences, but it does not draw any conclusions about the results that the user did not click on. As such, SpyNB is still valid even if the user does not follow any order in reading the search results or does not click on all relevant results. Our extensive offline experiments demonstrate that SpyNB discovers many more accurate preferences than existing algorithms do. The interactive online experiments further confirm that SpyNB and our personalization approach are effective in practice. We also show that the efficiency of SpyNB is comparable to existing simple preference mining algorithms.

Categories and Subject Descriptors: H.3.5 [**INFORMATION STORAGE AND RETRIEVAL**]: On-line Information Services

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Personalization, Clickthrough Data, Search Engine, User Preferences

1. INTRODUCTION

As the amount of information on the Web (World Wide Web) is abundant and personal electronic devices are ubiquitous, there has been much research work related to personalization with the objective to satisfy users' diversified needs in searching Web information [Liu et al. 2002; 2004; Jeh and Widom 2003; Haveliwala 2002; Sugiyama et al. 2004]. Most current search engines, however, return the same results to all users who ask the same query. This is clearly inadequate when the users have different search goals, tasks and interests. For example, for the query "apple", some users may be interested in Web pages about "apple" as a computer, while other users may want information related to "apple" as a fruit. In fact, current Web search engines return mostly pages about apple as a computer, making it difficult for users to retrieve pages about apple as a fruit. We can easily find many queries such as "mouse", "chair", "ir" and "Java", which may be interpreted by different users differently. We should also note that this problem is more than a problem of query semantics; even if a query is interpreted by users in the same way, users may still be looking for different aspects of the subject (e.g., one may be interested in Java tutorials while others may be interested in Java compilers). Therefore, delivering the same search results for the same query is not satisfactory. Recent work on search engine adaptation techniques aims to address this problem [Joachims 2002b; Tan et al. 2004; Deng et al. 2004].

In this paper, we tackle the problem of search engine adaptation by considering two main research issues. The first one is *preference mining*, which discovers user's preferences of search results from clickthrough data. For example, for a particular query, q, if a user chooses to click a search result, l_A, but skips another one, l_B, preference mining algorithms aim to discover the user's preferences from the clickthrough data, e.g., the user prefers search result l_A to l_B for query q. Clickthrough data (or we may simply say CT data) is a search engine log that records for each query the result list presented to the user as well as the links clicked on by the user. The second research issue is *ranking function optimization*, which optimizes the ranking (retrieval) function of a search engine according to the user's preferences.

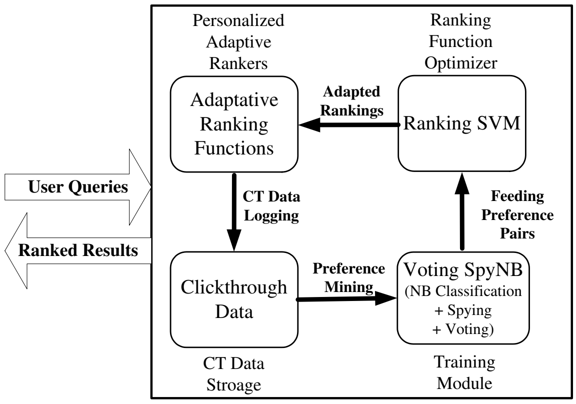

Fig. 1. The general process of search engine adaptation using clickthrough data

The general process of search engine adaptation is shown in Figure 1. The main idea is to use our new approach, SpyNB (Spy Naïve Bayes), to generate a set of preferences that are then fed into the RSVM (Ranking Support Vector Machine) algorithm for optimizing the ranking function for the user, which will be detailed in Sec 2.2. Essentially, SpyNB discovers the fragment preference pairs as constraints that are fed into the RSVM framework as shown in Figure 2. SpyNB is an effective means to generate the positive and negative datasets, from which accurate preference fragment pairs can be derived for optimizing the ranking function. In addition, the generated preference pairs do not rely on the strict scan order assumption. This approach also solves the problem that a user might skip some relevant links when he or she scans down the result list, leading to the extraction of wrong preference pairs.

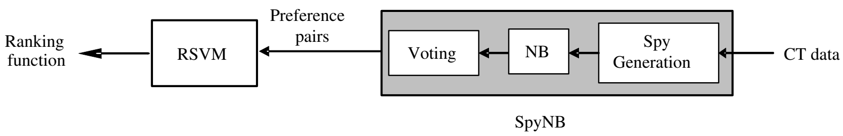

Fig. 2. A functional diagram of the SpyNB process

At the very beginning of search engine adaptation, an adaptable search engine adopts a general (not adapted) ranking function to serve a new user. Then, the user submits queries and clicks on the search results while the search engine logs the user's actions as clickthrough data for analysis. The clickthrough data is first processed by a preference mining algorithm, which outputs explicit user preferences in the form of "the user prefers l_A to l_B". Later a ranking function optimizer takes the explicit user preferences as input data and produces an optimized ranking function with respect to the user's preferences. Finally, the updated ranking function replaces the old general ranking function to serve the future queries of this particular user. At this stage, a round of search engine adaptation is finished. The adaptation process can be repeated regularly to determine the most updated user preferences.

It is worth mentioning that in our recent survey [Ke et al. 2005], we classified search engine adaptation into three categories, namely, *content-based personalization*, *link-based personalization* and *function-based personalization*. Our current approach falls in the third category. Essentially, we propose a new preference mining algorithm and extend the work of search engine adaptation to personalization, which is achieved through adapting the search engine's ranking function for individual users. In particular, our clickthrough interpretation is more reasonable and intuitive than previous approaches, since our preference mining algorithm does not make strong assumptions on how users read the search results.

The information source we investigate is clickthrough data, which can be formally represented as a triplet (q, r, c) [Joachims 2002b], where q is the input query, r is the result list of links $(l_1, \ldots, l_n)$, and c is the set of links that the user has clicked on. Figure 3 illustrates an example of clickthrough data for the query "apple". In the figure, the three links, l_1, l_4 and l_8, are in bold, indicating that they have been clicked on by the user. The advantage of using clickthrough data to discover a user's preferences is that it does not intervene the user's interaction with the searching process. The data can be collected by a search engine without additional burden on the user. Thus, clickthrough data are much easier to collect and more abundant than explicit feedback [Bartell et al. 1994] that requires the user's explicit ratings. However, the user's preferences conveyed by clickthrough data are *implicit* and sometimes ambiguous. Therefore, discovering the real user preferences from clickthrough data is non-trivial but critical to high-quality search engine adaptation. The reason is that if the identified preferences are inaccurate, optimizing the ranking function using inaccurate preferences can make the ranking (retrieval) quality worse.

Preference mining is a challenging problem as evidenced by the recent work in [Joachims 2002b; Deng et al. 2004; Joachims et al. 2005]. Existing algorithms are based on some strong assumptions on how users scan the search results in a strict order and then deduce the relative preferences, which may not be correct in reality. For example, Joachims' algorithm assumes that users scan search results strictly from top to bottom. However, it is possible that a user skips several results without examining them carefully. As a result, Joachims' assumption is too simplistic to predict all correct preference pairs to accurately reflect users' needs. We do not make this strong assumption about a user's scanning behavior but introduce a new

Links	The list of search results with titles, abstracts and URLs of Web pages
$\mathbf{l_1}$ **(clicked)**	**Apple** **Opportunities at Apple. Visit other Apple sites ...** ***http://www.apple.com/***
l_2	Apple - QuickTime - Download Visit the Apple Store online or at retail locations ... *http://www.apple.com/quicktime/download/*
l_3	Apple - Fruit Apples have a rounded shape with a depression at the top ... *http://www.hort.purdue.edu/ext/senior/fruits/apple1.htm*
$\mathbf{l_4}$ **(clicked)**	**Apple .Mac Welcome** **... member specials throughout the year. See ...** ***http://www.mac.com/***
l_5	www.apple-history.com A brief history of the company that changed the computing world ... *http://www.apple-history.com/*
l_6	MacCentral: Apple Macintosh News Steve Jobs unveils Apple mini stores. ... *http://www.macworld.com/news/*
l_7	Adams County Nursery, apple trees One of the most widely planted apple cultivars worldwide. *http://www.acnursery.com/apples.htm*
$\mathbf{l_8}$ **(clicked)**	**Apple - Support** **Support for most Apple products provided by Apple Computer** ***http://www.info.apple.com/***
l_9	AppleInsider ... Apple seeds Mac OS X Server 10.3.6 build $7R20$. *http://www.appleinsider.com/*
l_{10}	ROSE APPLE Fruit Facts The rose apple is too large to make a suitable container plant. ... *http://www.crfg.org/pubs/ff/roseapple.html*

Fig. 3. Search on the query “apple” and the CT data. (Links in bold are clicked on by the user.)

interpretation on clickthrough data based on the simple but reasonable assumption that the user’s preferences can be reflected by the links he or she clicks on. We do not make any explicit assumptions on the relevancy of the links that he or she did not click on. Accordingly, we propose a novel *Spy Naïve Bayes* algorithm for discovering preferences, denoted as SpyNB. Furthermore, we present an approach to personalizing a search engine through adapting its ranking function using SpyNB with a ranking function optimizer.

To evaluate the effectiveness of SpyNB and our search engine personalization approach, we personalize a metasearch engine that comprises MSNSearch [MSN], Overture [Ove] and WiseNut [Wis] in the experiments. The offline empirical results demonstrate that SpyNB discovers much more accurate preferences than Joachims' [Joachims 2002b] and mJoachims' [Deng et al. 2004] algorithms do. Moreover, we show that the ranking (retrieval) function personalized with SpyNB improves the ranking quality in terms of *the average rank of user's clicks* by 20% compared with the case without learning, which clearly indicates that the personalization effect is significant. Our interactive online experiments further confirm that the metasearcher personalized by SpyNB is significantly better in retrieval quality than MSNSearch and the metasearcher based on Joachims' algorithm.

In summary, this paper makes two main contributions. First, a novel SpyNB preference mining algorithm is proposed, which is demonstrated to be more effective and accurate than existing algorithms. Second, a search engine personalization framework based on preference mining is presented.

The rest of this paper is organized as follows: Section 2 surveys the related work. In Section 3, we introduce a new clickthrough interpretation. In Section 4, we present our SpyNB preference mining algorithm. In Section 5, we report empirical results when SpyNB applied to search engine personalization. Finally, Section 6 concludes the paper.

2. RELATED WORK

Personalization techniques have been developed in diversified ways (cf. see Section 5.1 of [Ke et al. 2005] for a detailed analysis). For example, content-based personalization deals with the "relevance" measure of Web pages and the user's queries. In this approach, the user's query is modified to adapt the search results for the specific user. In order to manage users' interests, a content-based personalization technique is used to construct users' profiles, which store users' interests derived from their search histories.

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 on 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. For example, a user may wish to use his or her bookmarks as a set of preferred pages, so that any retrieved pages that are important with respect to the bookmarked pages would be ranked higher than other non-bookmarked pages. It is worth mentioning that [Pretschner and Gauch 1999] introduced an ontology-based Web site mapping approach for identifying conceptual meta-information from local sites. The information can be used to classify Web pages into categories, which is an effective text classification approach for matching user preferences. The work in [Heer and Chi 2002] incorporated text analysis to discover preferences in order to obtain personalised ranking functions.

Research on personalizing search engines based on clickthrough consists of two main research issues: preference mining and ranking function optimization. A preference mining algorithm first discovers user's preferences on the search results from clickthrough data. A ranking function optimization method optimizes a search engine's ranking function according to the discovered preferences. We now review these two research issues in more detail in the following subsections, since they are directly relevant to our subsequent discussion.

2.1 Preference Mining Algorithms

Preference mining has been investigated in recent years. The mathematical foundation for preferences was studied in [Kießling 2002; Agrawal and Wimmers 2000]. In this paper, we adopt the *strict partial order* model [Kießling 2002] to express preferences.

DEFINITION 1. **(Preference)** *Given two retrieved links, l_i and l_j, for a given query, q, the pairwise preference, $l_i <_q l_j$, means that the user prefers l_j to l_i with respect to the query q.*

There are two existing algorithms for mining preferences from clickthrough data. One is the algorithm proposed in [Joachims 2002b], which assumes that the user scans the ranked list of the search results *strictly* from top to bottom. In particular, Joachims' algorithm elicits preferences based on a clickthrough interpretation as described in Interpretation 1. We hereafter refer to Joachims' algorithm in [Joachims 2002b] as "Joachims' algorithm" or simply "Joachims".

INTERPRETATION 1. *When a user scans the ranked list of the search results with respect to the query, q, if he or she does not click on a link, l_i, but clicks on a lower link, l_j, where $j > i$, then this indicates that the user prefers link l_j to l_i. In this case, the preference is identified by the partial order, $<_q$, and is denoted as $l_i <_q l_j$. The rationale is that when the user scans the search results from top to bottom, he or she must have observed l_i and decided to skip it, before he or she clicks on l_j.*

To exemplify Joachims' algorithm, consider the clickthrough example in Figure 3. According to Interpretation 1, all the preferences identified by Joachims' algorithm are shown in Table I.

Preferences containing l_1	Preferences containing l_4	Preferences containing l_8
Empty Set	$l_2 <_q l_4$	$l_2 <_q l_8$
	$l_3 <_q l_4$	$l_3 <_q l_8$
		$l_5 <_q l_8$
		$l_6 <_q l_8$
		$l_7 <_q l_8$

Table I. Pairwise preferences identified by Joachims' algorithm from the clickthrough data shown in Figure 3

Joachims' algorithm has been shown to have the problem of penalizing high-ranking links [Deng et al. 2004], which means that the high-ranking links (e.g., l_1, l_2) are *more* likely to be *"less preferred"* compared to the low-ranking links (e.g., l_9, l_{10}). Consider the preference example shown in Table I. Links l_1 and l_8 are both clicked links; however l_1 appears on the *right*-hand side of the preferences (meaning they are "preferred" by the user) *less* often than l_8 does (l_1, 0 times; l_8, five times). On the other hand, links l_2 and l_9 are both unclicked links; however, l_2 appears on the *left*-hand side of the preferences (meaning "not preferred" by the user) *more* often than l_9 does (l_2, twice; l_9, 0 times). This explains the problem of over-penalizing the high-ranking links.

To address the above problem, the *mJoachims' algorithm* [Deng et al. 2004] was proposed. We hereafter refer to mJoachims' algorithm as *mJoachims*. Besides Interpretation 1 of Joachims' algorithm, mJoachims further introduces Interpretation 2 in order to alleviate Joachims' problem with penalizing high-ranking links.

INTERPRETATION 2. *Suppose l_i is a clicked link, l_j is the next clicked link right after l_i (i.e., no other clicked links between l_i and l_j), and l_k is any unclicked link between l_i and l_j ($i < k < j$). When the user clicks on l_j, he or she must have observed link l_k ($k < j$) and decided not to click on it. Therefore, besides Interpretation 1, the clickthrough also indicates that the user prefers link l_i to l_k. Thus, the additional preferences $l_k <_q l_i$ can be identified.*

Overall, the preferences identified by mJoachims are those identified by the standard Joachims' algorithm plus the preferences $l_k <_q l_i$ ($i < k < j$). Consider again the clickthrough example in Figure 3. The pairwise preferences identified by mJoachims are shown in Table II. By comparing the preferences in Table I and Table II, we can see that mJoachims adds some preferences to the standard Joachims' algorithm with high-ranking links (e.g., l_1 and l_4) being the *preferred* links.

Preferences containing l_1	Preferences containing l_4	Preferences containing l_8
$l_2 <_q l_1$	$l_2 <_q l_4$	$l_2 <_q l_8$
$l_3 <_q l_1$	$l_3 <_q l_4$	$l_3 <_q l_8$
	$l_5 <_q l_4$	$l_5 <_q l_8$
	$l_6 <_q l_4$	$l_6 <_q l_8$
	$l_7 <_q l_4$	$l_7 <_q l_8$

Table II. Pairwise preferences identified by mJoachims' algorithm from the clickthrough data shown in Figure 3

2.2 Ranking Function Optimization

After the preferences have been discovered, a ranking function optimizer can take the preferences as input data to optimize the ranking function of a search engine. Joachims [Joachims 2002b] first proposed a *ranking SVM* algorithm, which solves the optimization problem using an SVM approach. Later, Tan *et al.* extended the ranking SVM using a co-training framework [Blum and Mitchell 1998] and proposed the *RSCF (Ranking SVM in Co-training Framework) algorithm*, which was reported to be better than the standard ranking SVM for small training data sets [Tan et al. 2004]. As the ranking SVM is used in our search engine adaptation experiments, we briefly revisit its main ideas in this section. For more details about the ranking SVM, readers may refer to Joachims paper [Joachims 2002b].

We now illustrate the basic idea of the ranking SVM by using a simple example shown in Figure 4. Suppose there are three links, l_1, l_2, and l_3, in the feature space and the input preferences are $l_3 <_q l_2 <_q l_1$. Let us compare two possible linear ranking functions, $\overrightarrow{\omega_1}$ and $\overrightarrow{\omega_2}$. (The formal definition of $\overrightarrow{\omega}$ and the feature space will be detailed in Section 5.2.) Note that the ranking result is equal to the order of the links projected on $\overrightarrow{\omega_1}$ and $\overrightarrow{\omega_2}$. As the figure shows, $\overrightarrow{\omega_1}$ ranks the three links as $l_3 <_q l_2 <_q l_1$, which is equivalent to the input preferences; while $\overrightarrow{\omega_2}$ ranks the links as $l_3 <_q l_1 <_q l_2$, which does not conform to all of the input preferences. Therefore, $\overrightarrow{\omega_1}$ is better than $\overrightarrow{\omega_2}$ for holding the input preferences. Moreover, if more than one ranking function can hold the input preferences, the one that maximizes the distance (marked as δ in the figure) between the two closest projections is the best. In the figure, $\overrightarrow{\omega_1}$ is the best ranking function, because it holds all the input preferences and also maximizes the distance, δ_1. The ranking SVM algorithm aims at finding the best ranking functions such as $\overrightarrow{\omega_1}$ in the example. For a large set of input preferences, ranking functions that hold all preferences may not exist. Then, the ranking SVM outputs a ranking function that holds as many preferences as possible.

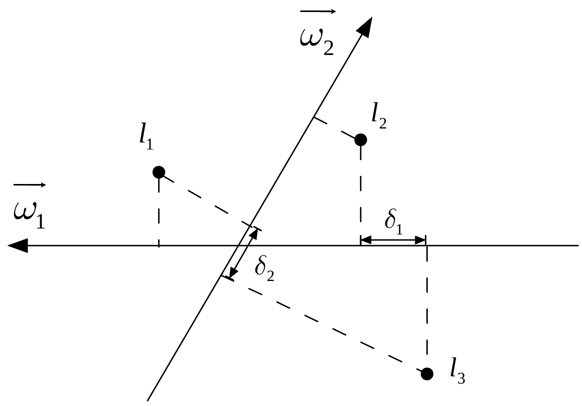

Fig. 4. Ranking links l_1, l_2, l_3 with functions $\overrightarrow{\omega_1}$ and $\overrightarrow{\omega_2}$

Beyond the simple example of the ranking SVM, we further describe its technique formally. Let q_k denote a query, D_k denote the set of retrieved documents of q_k, and P_k denote the set of discovered preferences from D_k: $P_k = \{d_i <_{q_k} d_j\}, \quad d_i, d_j \in D_k$. Given the training data:

$$T = \{(D_1, P_1), (D_2, P_2), \ldots, (D_n, P_n)\},$$

the ranking SVM aims at finding a ranking function, $f(q, d)$, which holds as many preferences in T as possible. $f(q, d)$ is defined as $f(q, d) = \overrightarrow{\omega} \cdot \phi(q, d)$, where $\phi(q, d)$ is a *feature vector* representing how well a document, d, matches a query, q, and $\overrightarrow{\omega}$ is a *weight vector*, which actually determines the ranking function, $f(q, d)$.

Thus, the problem of the ranking SVM becomes finding a $\overrightarrow{\omega}$ that holds the maximum number of the following inequalities:

$$\begin{aligned} \text{For all} \quad & d_i <_{q_k} d_j \in P_k \quad , \quad (1 \le k \le n) \\ & \overrightarrow{\omega} \cdot \phi(q_k, d_j) \; > \; \overrightarrow{\omega} \cdot \phi(q_k, d_i). \end{aligned} \tag{1}$$

The problem of solving $\overrightarrow{\omega}$ with the constraints in Equation (1) is *NP-hard* [Hoffgen et al. 1995]. An approximate solution can be obtained by introducing non-negative *slack variables*, ξ_{ijk}, to the inequalities to tolerate some ranking errors. The inequalities are rewritten as:

$$\begin{array}{ll} \text{For all} \quad (d_i <_{q_k} d_j) \in P_k \ , \quad (1 \le k \le n) \\ \overrightarrow{\omega} \cdot \phi(q_k, d_j) > \overrightarrow{\omega} \cdot \phi(q_k, d_i) + 1 - \xi_{ijk}, \quad \xi_{ijk} \ge 0, \end{array} \tag{2}$$

and the ranking SVM is then formulated as a constrained optimization problem, which is stated as minimizing the target function:

$$V(\overrightarrow{\omega}, \xi) = \frac{1}{2}\overrightarrow{\omega} \cdot \overrightarrow{\omega} + C \sum \xi_{ijk}, \tag{3}$$

subject to the constraints given in Equation (2).

The basic idea of solving the above optimization problem is as follows. Let δ be the distance between the two closest projected documents along a weight vector. For example, in Figure 4, δ_1 and δ_2 are the distances between the two closest projections along $\overrightarrow{\omega_1}$ and $\overrightarrow{\omega_2}$, respectively. If there are several weight vectors that are able to hold all rankings subject to the condition in Equation (2), the one that maximizes the margin, δ, is preferred. This is because the larger the value of δ, the more definite the ranking, and hence the better the quality of the weight vector, $\overrightarrow{\omega}$. The summation term, $\sum \xi_{ijk}$, of the slack variables in the target function (3) is the sum of the errors in the ranking pairs. Therefore, minimizing this summation term can be viewed as minimizing the *overall* training errors. Finally, parameter C is introduced to allow a trade-off between the margin size, δ, and the overall training error.

The ranking SVM returns as output a weight vector, $\overrightarrow{\omega}$, which is used to rank search results according to the value: $f(q, d) = \overrightarrow{\omega} \cdot \phi(q, d)$.

3. CLICKTHROUGH INTERPRETATION

In this section, we first discuss the inadequacy of the existing preference mining algorithms. Then, we introduce a new clickthrough interpretation for preference mining that does not rely on the user's scan order on the result list.

3.1 Inadequacy of Existing Algorithms

Although Joachims and mJoachims are simple and efficient, their extraction of preference pairs resulting from the strict scan order assumption may not be entirely correct. This is because, in reality, the user's behavior may be very diversified. For example, Joachims assumes that the user scans the search results strictly from top to bottom. However, it is possible that a user skips several results without examining them carefully and clicks on a link at a lower rank. However, both Joachims and mJoachims would conclude that these skipped links are uninteresting to the user but in fact we could only say that whether these links are interesting to the user or not is unknown. As a result, the preferences identified by Joachims and mJoachims may not reflect users' preferences accurately.

Let us consider again the clickthrough example for the query "apple" in Figure 3. After analyzing the titles, abstracts and URLs of all the ten links, we find that basically the links are about two different topics: links l_3, l_7 and l_{10} are about "apple fruit", while the other seven links are related to "Apple computer". Furthermore, we can see that the clicked links l_1, l_4 and l_8 (in bold) are all about "Apple computer". Therefore, an intuitive interpretation of this clickthrough data is that the user is looking for results about "Apple computer". From a preference mining point of view, we can infer that *the user likes links about "Apple computer" more than links about "apple fruit"*. Now, according to this interpretation, we list in Table III the real preferences conveyed by the clickthrough example. If the results of Table III are compared to those in Table I and Table II, we can see that the preferences identified by Joachims and mJoachims are not entirely accurate.

Preferences containing l_1	Preferences containing l_4	Preferences containing l_8
$l_3 <_q l_1$	$l_3 <_q l_4$	$l_3 <_q l_8$
$l_7 <_q l_1$	$l_7 <_q l_4$	$l_7 <_q l_8$
$l_{10} <_q l_1$	$l_{10} <_q l_4$	$l_{10} <_q l_8$

Table III. The real preferences of the clickthrough data shown in Figure 3

In the above example, the problem of the existing algorithms is that they mistakenly identify some high-ranking unclicked links about "Apple computer" (e.g., l_2, l_5) as "unpreferred" links. We argue that in practice it is possible that the user does not click on all of the links relevant to his or her interests, because he or she may not be patient enough to examine all the relevant links, or he or she may stop clicking after seeing "enough" information, and thus leave some relevant links unclicked. Moreover, a user may skip a relevant link because the abstract of that link is not informative enough. However, existing algorithms cannot handle the above-mentioned possibilities but simply derive preferences based on the simple rule that *if a high-ranking link is not clicked, it is then considered as an "unpreferred" link.*

3.2 New Clickthrough Interpretation

Motivated by the example in Section 3.1, we aim to design an algorithm that can find the exact preferences in Table III based on the clickthrough data in Figure 3 in an effective way.

We note that a user typically judges the links based on the summaries[1] displayed on the result page and clicks on the links that appear to be of interesting to him or her. Therefore, it is reasonable to assume that the clicked links collectively reflect the user's preferences. Moreover as stated before, the user is unlikely to click on all of the returned links that match his or her interests. Thus, it is also reasonable to assume that the unclicked links consist of links that the user may or may not prefer. We then assume that the links not preferred by the user are those with topics different from that of the clicked links. For example, if the search results are on three topics A, B and C, when the user clicks on links that are relevant only to

[1]Most search engines display textual information such as titles and abstracts, in addition to non-textual information such as last modification dates, size, etc.

A, we can treat B and C as unpreferred topics; when the user clicks on links that are about topics A and B, then C is treated as unpreferred.

Formally, our clickthrough interpretation is described as follows.

Interpretation 3 (Our interpretation). *We treat the links clicked by the user as* positive *examples and those not clicked as* unlabeled *data. Let P denote the positive set, and U denote the unlabeled set. Then, by analyzing the textual summaries, we can identify which links in U are on a different topic than that of the positive links and take them as the* predicted negative *examples. Let PN denote the predicated negative set ($PN \subset U$). Then, the clickthrough data indicate that the user likes all the links in P better than all the links in PN. The preferences are expressed as follows:*

$$l_j <_q l_i, \qquad \forall\ l_i \in P, \quad l_j \in PN. \tag{4}$$

According to Interpretation 3, the preferences conveyed by the clickthrough in Figure 3 are those listed in Table III. Remarkably, our interpretation does not assume how the user scans the search results, but only assumes that *the links "preferred" by the user and the links "unpreferred" by the user are about different topics*. We believe that this assumption is reasonable and reflects user behaviors. Moreover, our idea of analyzing the texts (e.g., titles and abstracts) of the links for discovering preferences is reasonable, since it is generally believed that users read the summaries to judge if a link is relevant to their information needs.

4. SPY NAÏVE BAYES

In this section, we propose a new preference mining algorithm, called *Spy Naïve Bayes* (SpyNB). It consists of two main components: a spying technique to obtain more accurate negative samples and a voting procedure to consider the opinions of all spies.

According to our clickthrough interpretation, we need to categorize *unlabeled* data in order to discover the *predicted negative* links. Naïve Bayes [Mitchell 1997] is a simple and efficient text categorization method. However, conventional Naïve Bayes requires both positive and negative examples as training data, while we only have positive examples. To address this problem, we employ a spying technique [Liu et al. 2003; Liu et al. 2002] to train Naïve Bayes by incorporating unlabeled training examples. Moreover, in order to obtain more accurate *predicted negatives*, we further introduce a voting procedure to make full use of all potential spies. Finally, we propose our *Spy Naïve Bayes* algorithm.

4.1 The Spying Technique and Voting Procedure

We first describe how the standard Naïve Bayes is adapted in our context as follows. Let "+" and "–" denote the positive and negative classes, respectively. Let $L = \{l_1, l_2, \ldots, l_N\}$ denote a set of N retrieved links. Each link, l_i, is represented as a word vector, $W = (w_1, w_2, \ldots, w_M)$, where we keep the number of occurrences of w_i appearing in the summary. Then, Naïve Bayes can be trained by estimating the prior probabilities ($Pr(+)$ and $Pr(-)$), and likelihood ($Pr(w_j|+)$ and $Pr(w_j|-)$) as shown in Algorithm 1. It is also straightforward to observe that $Pr(+) = (1 - Pr(-))$.

Algorithm 1 Training the Naïve Bayes Algorithm

Input:

$L = \{l_1, l_2, \ldots, l_N\}$ /* a set of links */

Output:

Prior probabilities: $Pr(+)$ and $Pr(-)$;

Likelihoods: $Pr(w_j|+)$ and $Pr(w_j|-)$ $\forall j \in \{1, \ldots, M\}$

Procedure:

1: $Pr(+) = \frac{\sum_{i=1}^{N} \delta(+|l_i)}{N}$;

2: $Pr(-) = \frac{\sum_{i=1}^{N} \delta(-|l_i)}{N}$;

3: **for** each attribute $w_j \in W$ **do**

4: $\quad Pr(w_j|+) = \frac{\lambda + \sum_{i=1}^{N} Num(w_j, l_i)\delta(+|l_i)}{\lambda M + \sum_{k=1}^{M} \sum_{i=1}^{N} Num(w_k, l_i)\delta(+|l_i)}$;

5: $\quad Pr(w_j|-) = \frac{\lambda + \sum_{i=1}^{N} Num(w_j, l_i)\delta(-|l_i)}{\lambda M + \sum_{k=1}^{M} \sum_{i=1}^{N} Num(w_k, l_i)\delta(-|l_i)}$;

6: **end for**

In Algorithm 1, $\delta(+|l_i)$ indicates the class label of link l_i. Its value is 1 if l_i is positive and 0 otherwise. $Num(w_j, l_i)$ is a function counting the number of times w_j appears in l_i. λ is a smoothing factor [McCallum and Nigam 1998]; we set $\lambda = 1$ to make Naïve Bayes more robust.

When predicting unlabeled links, Naïve Bayes calculates the *posterior probability* of a link, l, using the Bayes rule:

$$Pr(+|l) = \frac{Pr(l|+)Pr(+)}{Pr(l)},$$

where $Pr(l|+) = \prod_{j=1}^{|w_l|} Pr(w_{l_j}|+)$ is the product of the likelihoods of the keywords in link l. Then, link l is predicted to belong to class "+", if $Pr(+|l)$ is larger than $Pr(-|l)$ and "–" otherwise.

When the training data contains only positive and unlabeled examples, the spying technique can be introduced to learn the Naïve Bayes classifier. The idea behind the procedure is illustrated in Figure 5. First, a set of positive examples, S, are randomly selected from P and put in U to act as "spies". Then, the unlabeled examples in U together with S are regarded as negative examples to train the Naïve Bayes classifier. The trained classifier is then used to assign *posterior probability*, $Pr(+|l)$, to each example in $(U \cup S)$. After that, a threshold, T_s, is determined based on the *posterior probabilities* assigned to S. An unlabeled example in U is selected as a predicted negative example if its probability is less than T_s. The examples in S act as "spies", since they are positive and put into U pretending to be negative examples. During the process of prediction, the unknown positive examples in U are assumed to have similar behavior as the spies (i.e., assigned comparative probabilities). Therefore, the predicted negatives, PN_i, can be identified, which is separated from U. As a result, the original U is split into two parts after the training. One is PN_i which may still contain some positive items (white region) due to error in the classification arising from p_i. Another is the remaining items in U which may still contain some negative items (black region), also due to error in the classification. Note that p_i returns to P, since it is known to be (sure) positive.

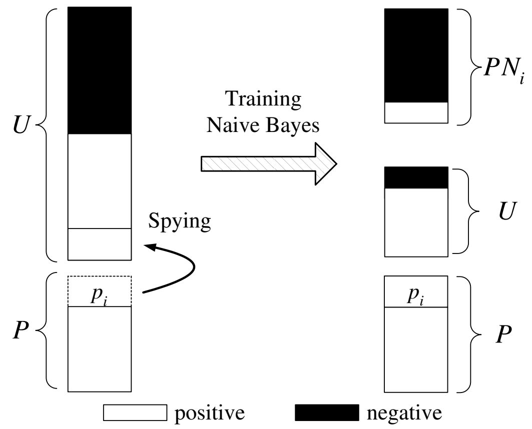

Fig. 5. The underlying principle of the spying technique

We notice that, in our spying technique, the identified PN can be influenced by the selection of spies. As for clickthrough data, there are typically very few positive examples (recall that they are clicked links). We can make full use of all the potential spies to reduce the influence. Thus, we introduce a voting procedure to strengthen the spying technique further.

The idea of a voting procedure is depicted in Figure 6 and is explained as follows. First of all, the algorithm runs the spying technique n times, where $n = |P|$ is the number of positive examples. Each time, a positive example, p_i, in P is selected to act as a spy and put into U to train the Naïve Bayes classifier, NB_i. The probability, $Pr(+|p_i)$, assigned to the spy, p_i, can be used as the threshold, T_s, to select a candidate predicted negative set (PN_i). That is, any unlabeled example, u_j, with a smaller probability of being a positive example than the spy ($Pr(+|u_j) < T_s$) is selected into PN_i. As a result, n candidate predicted negative sets, $PN_1, PN_2, \ldots, PN_n$, are identified. Finally, a voting procedure is used to combine all PN_i into the final PN. An unlabeled example is included in the final PN, if and only if it appears in at least a certain number (T_v) of PN_i. T_v is called the *voting threshold*. The voting procedure selects PNs based on the opinions of all spies and thus minimizes the bias of the spy selection.

4.2 The SpyNB algorithm

We now present the *Spy Naïve Bayes* algorithm in Algorithm 2. In the SpyNB algorithm, Steps 2 to 15 employ the spying technique $|P|$ times to generate $|P|$ candidate sets of PN_i. Steps 16 to 21 combine all PN_i into the final PN using spy voting.

To analyze the time complexity of SpyNB, we let $|P|$ denote the number of clicked links (positive examples), $|U|$ denote the number of unclicked links (unlabeled examples) and N denote the number of all links. Training Naïve Bayes (Algorithm 1) requires only one scan of all links. Thus, the time complexity of training is $O(N)$. The prediction of Naïve Bayes costs $O(|U|)$ time, where $|U| < N$. Thus, Steps 2 to 15 of SpyNB cost $O(|P| \cdot (N + |U|)) = O(|P| \cdot N)$ time. With a similar analysis, the

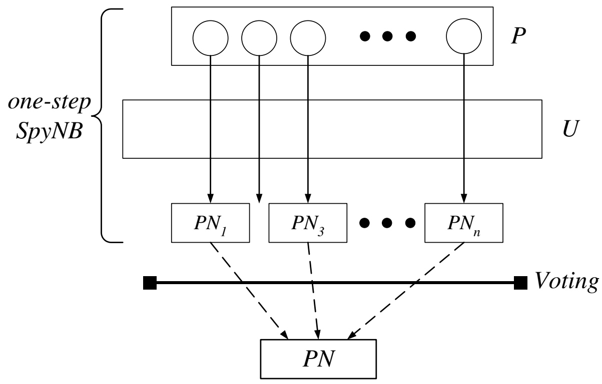

Fig. 6. The voting procedure

Algorithm 2 The Spy Naïve Bayes (SpyNB) Algorithm

Input:

P – a set of positive examples; U – a set of unlabeled examples; T_v – a voting threshold;

Output:

PN – the set of predicted negative examples

Procedure:

1: $PN_1 = PN_2 = \cdots = PN_{|P|} = \{\}$ and $PN = \{\}$;
2: **for** each example $p_i \in P$ **do**
3: $\quad P_s = P - \{p_i\}$;
4: $\quad U_s = U \cup \{p_i\}$;
5: $\quad$ Assign each example in P_s the class label 1;
6: $\quad$ Assign each example in U_s the class label -1;
7: $\quad$ Train a Naïve Bayes on P_s and U_s using Algorithm 1;
8: $\quad$ Predict each example in U_s using trained Naïve Bayes;
9: $\quad$ Spy threshold $T_s = Pr(+|p_i)$;
10: $\quad$ **for** each $u_j \in U$ **do**
11: $\quad\quad$ **if** $Pr(+|u_j) < T_s$ **then**
12: $\quad\quad\quad PN_i = PN_i \cup \{u_j\}$;
13: $\quad\quad$ **end if**
14: $\quad$ **end for**
15: **end for**
16: **for** each $u_j \in U$ **do**
17: $\quad Votes =$ the number of PN_i such that $u_j \in PN_i$
18: $\quad$ **if** $Votes > T_v \cdot |P|$ **then**
19: $\quad\quad PN = PN \cup \{u_j\}$;
20: $\quad$ **end if**
21: **end for**

---

time complexity of Steps 16 to 21 of SpyNB is $O(|P| \cdot |U|)$, which is smaller than $O(|P| \cdot N)$.

Overall, the time complexity of SpyNB is $O(|P| \cdot N)$. We know that the time complexity of Joachims and mJoachims are both $O(N)$. Although, SpyNB is not as efficient as Joachims and mJoachims based on the complexity analysis, in practice $|P|$ is very small making SypNB's time complexity in effect constant bound. For example, the empirical clickthrough data reported in [Tan et al. 2004] shows that it has merely 2.94 clicks per query on average.

By employing SpyNB for mining preferences and ranking SVM for ranking function optimization, we are able to build a personalized ranking function by serving the user with the specific ranking function adapted with his or her clickthrough. In practice, to identify the user's ID, a search engine can use cookies or require the user to login before he or she uses the personalized search service.

5. EXPERIMENTAL EVALUATION

We conducted both offline and online experiments to evaluate the effectiveness of SpyNB and our search engine personalization approach. The ranking SVM used in our experiments was implemented with the SVM-Light package [Joachims 1999], which can be downloaded from [SVM].

5.1 Experimental Setup: Personalized Metasearch

In general, our personalization approach can be used to personalize a standalone search engine. However, in the experimental evaluation, we apply our personalization approach to a metasearch engine. There are some advantages of adopting a metasearch engine for experimental evaluation. First, the end users do not see any difference between a single search engine and a metasearch engine; in both cases, the users see a uniform list of results without knowing which search engine or metasearch engine they are using. Second, a metasearch engine allows us to choose different underlying search engines with different strengths, coverages and focuses, thus giving us an additional dimension on which to personalize the search results. Finally, a metasearch engine does not need to deal with crawling and indexing issues, which are not the goal of our paper.

Our metasearch engine comprises MSNSearch [MSN], WiseNut [Wis] and Overture [Ove]. At the time we conducted the experiments, MSNSearch was one of the most popular general search engines. WiseNut was a new and growing search engine. Overture was specialized in the advertising domain, which ranked results based on the prices paid by the sponsors. The three search engines have different strengths, coverages and focuses, and thus are suitable for us to evaluate the personalization effect.

We asked three groups of students from three different departments at our university, namely Computer Science, Finance and Social Science to use our metasearch engine. Each group had ten students. We assumed the following about our subjects: users from different departments have different interests but users within the same department share the same interests. The students from computer science are looking for computer science information; the finance students are interested in product information; and the social science students prefer to receive news. As far as the personalization method is concerned, the three groups of students can be considered as three "logical" users and the personalization methods tries to adapt the metasearch engine to deliver the best results to the respective group of users. Using more than one student in each group ensures that the experimental results are not affected by a few peculiar actions made by one or two users.

To collect the clickthrough data, each of the three groups of students submits to the metasearch engine 30 queries that are related to their interests. The metasearch engine at the beginning adopts a default ranking function to deliver results. The default ranking function combines the retrieved results from the underlying search engines in a *round-robin* manner. If a result is returned by more than one search engine, one of the results is randomly picked and presented only once. Moreover, all the links are displayed in a uniform format. Thus, a user cannot tell which search engine a result is from. These precautions ensured that we obtained unbiased clickthrough data. The same method was adopted in [Joachims 2002a]. Table IV shows some statistics of the clickthrough data we collected.

Departments	Computer Science	Social Science	Finance
Number of queries	300	300	300
Number of clicks	1230	875	1302
Avg. clicks per query	4.1	2.9	4.3
Avg. rank clicked on	5.87	5.6	5.59

Table IV. Statistics of our clickthrough data set.

5.2 Linear Ranking Function

Our metasearch engine adopts a linear ranking function to rank search results. Suppose q is a query and l is a link related to a Web document returned from the underlying search engines. The links are ranked according to the value $f(q,l) = \overrightarrow{\omega} \cdot \phi(q,l)$, where $\phi(q,l)$ is a feature vector representing the match between query q and link l, and $\overrightarrow{\omega}$ is a weight vector that can be learned by our adaptation approach. We then define the feature vector, $\phi(q,l)$, as three kinds of features, namely, *Rank Features*, *Common Features* and *Similarity Features*:

(1) **Rank Features** (3 numerical and 12 binary features).
Let $E \in \{M, W, O\}$ (M stands for MSNsearch, W for WiseNut, and O for Overture) and $T \in \{1, 3, 5, 10\}$ (the rank value). We define numerical features, *Rank_E*, and binary features, *Top_E_T*, of document d as follows:

$$Rank\_E = \begin{cases} \frac{11-X}{10} & \text{if document } d \text{ ranks at } X \text{ in} \\ & \text{the result of } E \text{, and } X <= 10; \\ 0 & \text{otherwise.} \end{cases}$$

$$Top\_E\_T = \begin{cases} 1 & \text{if } d \text{ ranks top } T \text{ in } E; \\ 0 & \text{otherwise.} \end{cases}$$

(2) **Common Features** (2 binary features).
—*Com_2*:
If the retrieved document ranks the top 10 in at least two search engines, the value is 1, otherwise it is 0.
—*Com_3*:
If the retrieved document ranks top 10 in three search engines, the value is 1, otherwise it is 0.

(3) **Similarity Features** (1 binary and 2 numerical features).
—The similarity between query and URL.

$$Sim\_U = \begin{cases} 1 & \text{if any word in } q \text{ appears in URL;} \\ 0 & \text{otherwise.} \end{cases}$$

—Sim_T:
The cosine similarity between query and title.
—Sim_A:
The cosine similarity between query and abstract.

Overall, $\phi(q, l)$ contains 20 features as shown below:

$$\begin{aligned} &(Rank\_M, Top\_M\_1, \ldots, Top\_M\_10, Rank\_W, \ldots, \\ &Rank\_O, \ldots, Com\_2, Com\_3, Sim\_U, \ldots, Sim\_A). \end{aligned} \tag{5}$$

Corresponding to the above feature vector definition, the weight vector, $\overrightarrow{\omega}$, contains 20 weights, each of which reflects the importance of a feature in Equation (5). Our definitions of $\phi(q, l)$ and $\overrightarrow{\omega}$ are defined in a similar way as those adopted in [Joachims 2002b; Tan et al. 2004].

5.3 Offline Experimental Analysis

The offline experiments consist of two parts. In the first part, we compare the effectiveness of SpyNB with Joachims and mJoachims on preference mining. Moreover, we evaluate if the ranking function personalized with SpyNB can improve the ranking quality of the original search results. In the second part, we analyse the effect of the voting threshold on the performance of SpyNB. We also make some interesting observations on the adaptive ranking function related to the strengths of the underlying search engines.

5.3.1 *Evaluation of Ranking Quality.* In order to compare SpyNB with other preference mining algorithms, we incorporate SpyNB, Joachims and mJoachims with ranking SVM to obtain three personalized ranking functions. We arbitrarily set the voting threshold of SpyNB (T_v in Algorithm 2) to 50%. Then, we rerank the original search results with the personalized ranking functions and see if they can improve the ranking quality.

Intuitively, a good ranking function should give high ranking to links that the users want. Thus, the smaller the average rank of the users' clicks, the better the ranking quality. According to this intuition, we measure ranking quality based on *the average rank of users' clicks*, denoted by Ψ. To show the actual improvement, we define a metric, *"relative average rank of users' clicks"*, denoted by Ψ^r, as the ratio of Ψ derived from a personalized ranking function divided by Ψ of the original search result. If $\Psi^r < 1$, then it indicates that an actual improvement is achieved.

The results are shown in Figure 7. First, the values of Ψ^r of SpyNB are all about 0.8, which means that the ranking function personalized with SpyNB satisfies the three user groups better than the original ranking does. Thus, the effect of personalization is significant. In particular, the improvement of SpyNB in ranking quality is about 20%, which clearly indicates that SpyNB is effective in preference mining. Moreover, we find that Joachims and mJoachims fail to achieve any actual

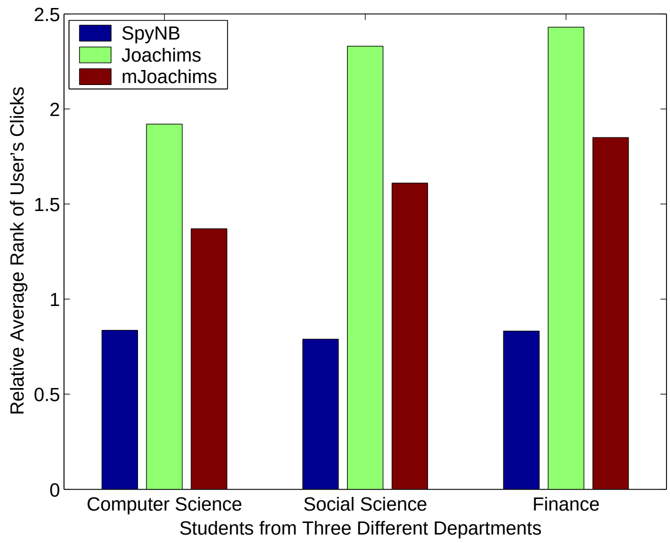

Fig. 7. Relative Average Rank of Users' Clicks of three preference mining algorithms

improvement after reranking the original search results, since their Ψ^r values are greater than 1. This can be attributed to their strong assumptions (recall Interpretations 1 and 2 in Section 2.1) that do not hold in our empirical clickthrough data. Thus, the preferences identified by the existing algorithms are incorrect. Specifically, mJoachims is better than Joachims, which can be attributed to Joachims penalty imposed on high-ranking links, while mJoachims alleviates this problem. Finally, we can conclude that the preferences discovered by SpyNB are much more accurate than those discovered by Joachims and mJoachims.

5.3.2 *Effect of Varying the Voting Threshold.* The voting threshold, T_v, in Algorithm 2, is the only parameter that a user needs to decide for SpyNB. In order to study the impact of T_v on the performance of SpyNB, we carried out an experiment to test various values of T_v. The result is presented in Figure 8.

As elaborated in Section 4.1, the T_v value reflects the confidence that SpyNB has in a single "spy" in selecting the *predicted negative* (PN) examples. On the one hand, small T_v values (e.g., 20%) imply that SpyNB is "credulous", since it may assign a link as an PN example based on the results of just one or two spies. On the other hand, large T_v values (e.g., 100%) mean that SpyNB is "conservative". In this case, it assigns a link as a PN if and only if all the spies decided that the link is a PN. Thus, the larger the value of T_v is, the more conservative SpyNB is, and the fewer *predicted negative* examples are selected.

Figure 8 shows that T_v indeed affects the performance of SpyNB, since the curves are sloped. The optimal values generally lie in the range of 20% to 40%. Large T_v values decrease the performance of SpyNB, indicating that large T_v values make SpyNB too conservative, which results in the missing of some real PN examples. On the other hand, overly small T_v values may have the problem of admitting noisy

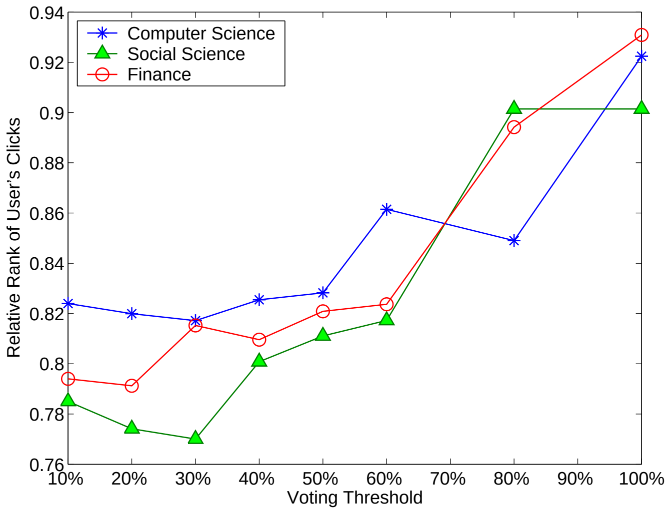

Fig. 8. Performance of SpyNB with varying voting threshold, T_v

PN examples, which can also be observed in Figure 8.

Finally, it is worth pointing out that the voting threshold gives important flexibility to SpyNB. We note that users in reality have diversified interests and behaviors. The voting threshold can be used to adapt SpyNB to different users. For example, in Figure 8, $T_v = 30\%$ is the optimal for the social science students, while $T_v = 40\%$ is the optimal for the finance students. Compared with the existing algorithms that are based on strong assumptions of the user's scanning behavior, SpyNB is more flexible in adapting to different users' preferences.

5.3.3 *Analysis of the Adapted Ranking Function.* As detailed in Section 5.2, the ranking function of our metasearch engine is composed of 20 weighted features. The adapted ranking function is examined in order to find out which features better reflect users' interests. We list two adapted ranking functions derived from the clickthrough of the computer science students (in Table V), and the finance students (in Table VI), respectively. Similar observations can also be found for the group of social science students, which are not presented here.

We have analyzed the meaning of each feature and weight. As detailed in Section 5.2, the ranking function is defined as $f(q, d) = \overrightarrow{\omega} \cdot \phi(q, d)$, which is the inner product of a feature vector, $\phi(q, d)$, and a weight vector, $\overrightarrow{\omega}$. Roughly speaking, features with high absolute weights have large impacts on the result ranking. In particular, the numerical *Rank Features*, *Rank_M*, *Rank_O* and *Rank_W*, reflect the relative importance of MSNSearch, Overture and WiseNut, respectively.

From Tables V and VI, we can observe that the weights of feature *Rank_M* are large for both groups of students; the weight of *Rank_O* is small for the computer science students, but large (almost equal to *Rank_M*) for the finance students; and the weight of *Rank_W* is moderate for the computer science students, but very

Feature	Weight	Feature	Weight
Rank_M	1.811	*Rank_W*	1.275
Top_M_1	0.566	*Top_W_1*	0.480
Top_M_3	-0.003	*Top_W_3*	0.229
Top_M_5	0.063	*Top_W_5*	-0.138
Top_M_10	-0.021	*Top_W_10*	-0.458
Rank_O	0.415	*Sim_A*	0.357
Top_O_1	-0.677	*Sim_T*	0.785
Top_O_3	0.447	*Sim_U*	0.288
Top_O_5	-0.087	*Com2*	0.186
Top_O_10	-0.440	*Com3*	-0.226

Table V. Adapted ranking function for computer science students

Feature	Weight	Feature	Weight
Rank_M	1.154	*Rank_W*	-0.217
Top_M_1	0.108	*Top_W_1*	0.355
Top_M_3	0.563	*Top_W_3*	0.362
Top_M_5	-0.045	*Top_W_5*	-0.364
Top_M_10	-0.757	*Top_W_10*	-1.429
Rank_O	1.019	*Sim_A*	0.025
Top_O_1	0.718	*Sim_T*	0.520
Top_O_3	0.586	*Sim_U*	-0.106
Top_O_5	0.528	*Com2*	0.240
Top_O_10	-0.864	*Com3*	0

Table VI. Adapted ranking function for finance students

small for the finance students.

It is interesting to note that these observations actually match the users' interests and the nature of the search engine components. For example, the fact that both the weights of *Rank_M* are large indicates that both groups of students like the results returned by MSNSearch. Since MSNSearch is widely considered as one of the best general search engines, it is not surprising to see that both groups of students like its results. As another example, we know that Overture is a search engine that specializes in advertising. Thus, it has a special strength in searching for *product information*, which matches the interests of finance students but not computer science students. The experimental results confirmed this intuition, since the value of *Rank_O* is large for finance students, but small for computer science students, which exactly matches our intuition. Roughly speaking, both groups of students prefer Overture's results to WiseNut's results. This is also reasonable, since WiseNut is a new and growing search engine that still needs to be improved. The *Rank_M* values for both groups of students are not so large, though MSMsearch seems to be a popular search engine.

As another interesting observation, we find that the values of *Sim_T* (the similarity between query and title) are larger than those of *Sim_A* (the similarity between query and abstract), meaning that users tend to select results with titles matching the query. Again, this result conforms to our intuition since the titles were created by the authors to precisely capture the page contents and they are displayed prominently on the result page.

We can also make other observations from the adapted ranking functions. Analyzing the functions is not only useful for observing the personalization effect but also for understanding the users' interests and behaviors.

5.4 Interactive Online Experiment

In order to verify that the ranking function personalized with SpyNB does improve retrieval quality in practice, we further asked the same three groups of students, who participated in our offline experiment, to conduct an interactive online evaluation. Again, each student submitted 30 queries, which were related to his or her interests.

The online experiment compares the three rankers: the ranker derived from SpyNB, that derived from Joachims and that of MSNSearch. The experimental procedure is as follows. When a user submits a query to our system, three rankings produced by the three rankers are obtained. We then combine the three rankings into an *unbiased* combined list using the same method of obtaining *unbiased* clickthrough data as described in Section 5.1. The property of *unbiased* combining is to ensure that the final ranking presented to the user is fair to all the sources. Finally, our system captures the new users' clickthrough data on the unbiased combined list.

We now explain how we evaluate the quality of different rankings. Let l be a clicked link in the combined list of query q, R_a and R_b are two rankings for comparison. Suppose that l is ranked as ith and jth in R_a and R_b, respectively. (If l is not in a ranking, its rank is set to a large enough number.) We say that the clicked link, l, favors ranking R_a if $i < j$, since it ranks higher in R_a than in R_b. After all the clicked links of query q are examined, we can conclude that R_a is better than R_b with respect to q, if there are more links favoring R_a than R_b.

For example, suppose two links, l_1 and l_2, in the combined result of query q are clicked; and link l_1 ranks 4th and 9th in rankings R_a and R_b, respectively, while link l_2 ranks the same, 5th and 5th, in both rankings. In this case, link l_1 favors R_a more than R_b, and l_2 favors both equally. Therefore, R_a is better than R_b for query q. Such evaluation not only takes into consideration the quantity of a ranking (the number of clicked links) but also the quality of a ranking (the ranks of clicked links).

In order to screen the online experimental results in different granularities, we further analyze the candidate rankings with different numbers of user's clicks per query. Specifically, we adopt a top-k parameter for screening, which means that, in each row in Tables VII and VIII, only the top-k clicks are counted. For example, if $k = 1$, then the top-1 parameter means that in this row, only the first click of the user is considered.

We present the comparison result of SpyNB with Joachims in Table VII and the result of SpyNB with MSNSearch in Table VIII. In both tables, "Tie" means that there were equal numbers of links favoring R_a and R_b. Moreover, as the largest number of user's clicks for a query is 8 for the data we collected in the online evaluation, we adopt four different values: 1, 3, 5 and "all", for the top-k parameter to present the result, in which "all" means that the comparison is based on all users' clicks (up to 8).

The online result clearly indicates that the result ranking derived from SpyNB

Comparison on Top-k clicks	R_a better than R_b	R_b better than R_a	Tie	No clicks	Total
1	63	15	2	10	90
3	61	15	8	6	90
5	57	14	16	3	90
All	59	17	12	2	90

Table VII. Comparison on the rankings of SpyNB (R_a) and that of Joachims (R_b)

Comparison on Top-k clicks	R_a better than R_b	R_b better than R_a	Tie	No clicks	Total
1	49	24	4	13	90
3	43	27	16	4	90
5	41	33	14	2	90
All	42	30	16	2	90

Table VIII. Comparison on the rankings of SpyNB (R_a) and that of MSNSearch (R_b)

is much better than the results derived from Joachims and MSNSearch, since the values in the first column are consistently larger than the values in the second column in both Tables VII and VIII.

We now further apply a *one-tailed binomial sign test* [Goulden 1956] on the observed data, in order to justify, in terms of statistics, how significant the superiorities of SpyNB are to Joachims and MSNSearch. The *binomial signed test* is a commonly used statistical hypothesis testing method when the observed data are binary. In our context, the observed data are either "R_a is better than R_b" or "R_b is better than R_a", which is binary, so that the *binomial signed test* is well suited. Specifically, we are going to test if it is statistically significant that "R_a is better than R_b" for both Tables VII and VIII. We let the *null hypothesis* be H_0: R_a and R_b are equally good, the *alternative hypothesis* be H_A: R_a is better than R_b, and the *level of significance* be α. The intuition of the test is that if the *null hypothesis* is true, then the difference of the observed values of "R_a is better than R_b" and "R_b is better than R_a" cannot be too large; otherwise the *null hypothesis* must be false. Technically, we need to compute a p-value for a pair of observed samples and check if p exceeds the critical value [Goulden 1956]. (The reader may refer to [Goulden 1956] for the detailed formulae.) We present the numerical results of the tests, which are computed with Matlab software, as given in Table IX. The basic idea is that p is viewed as the probability of wrongly rejecting the null hypothesis if it is in fact true. We thus reject the null hypothesis if the p-value is less than the level of significance α. We adopt the commonly used symbols to indicate the test result: a single asterisk ($\star$) if the null hypothesis is rejected at the 0.05 level of significance, which is standard requirement, and two asterisks ($\star\star$) if it is rejected at the 0.01 level, which is a stringent requirement.

The computed p-values and significance levels in Table IX show that the supe-

Engines Comparison	Top-k Clicks	p-value	Test Result
SpyNB *better than* Joachims	1	1.88×10^{-8}	$\star\star$
SpyNB *better than* Joachims	3	4.92×10^{-8}	$\star\star$
SpyNB *better than* Joachims	5	1.34×10^{-7}	$\star\star$
SpyNB *better than* Joachims	All	7.00×10^{-7}	$\star\star$
SpyNB *better than* MSNSearch	1	2.30×10^{-3}	$\star\star$
SpyNB *better than* MSNSearch	3	3.61×10^{-2}	$\star$
SpyNB *better than* MSNSearch	5	2.08×10^{-1}	
SpyNB *better than* MSNSearch	All	9.75×10^{-2}	

Table IX. The p-values and significance levels for the comparison result in Table VII and Table VIII

riority of the SpyNB ranker over the Joachims' ranker is consistently at a 99% significance level for any top-k clicks parameter, and the superiority of the SpyNB ranker over MSNSearch is at a significance level that varies from 75% ($p < 0.25$) to 99% ($p < 0.01$) depending on different values of the top-k clicks parameter. The superiority of the SpyNB ranker over the Joachims' ranker and MSNSearch are remarkable. The online results confirm again that SpyNB discovers more accurate preferences than the Joachims' algorithm. Furthermore, as MSNSearch is regarded as the strongest search engine component in our experiment, the superiority of SpyNB ranker over MSNSearch indicates that our personalized metasearch engine is better than its components. This verifies that our search engine personalization approach is effective in practice.

6. CONCLUSIONS AND FUTURE WORK

Personalization in Web search is an important research problem and is attracting a great deal of attention from the research community. We proposed a SpyNB preference mining algorithm, which is more effective and flexible than the existing algorithms. The contribution of SpyNB to preference mining is significant, since it is based on a new clickthrough interpretation and the application of the spying technique to ranking adaptation is a novel approach. Importantly, the interpretation does not assume any scanning order on the ranked results, which has been shown in this paper to perform much better than the existing methods. Our application of the spy voting procedure in adapting rankings is an interesting and novel approach. In the experiments, we personalized a metasearch engine using SpyNB. Both the offline and online results showed that our approach and algorithm are effective: the personalized metasearch engine improved the ranking quality and was able to cater for users' specific interests.

Admittedly, the very recent finding in [Joachims et al. 2005] suggests that there may be a "trust bias" effect on top links, which might restrict the accuracy of our classification. A solution to tackling this problem is to impose a weight on spies according to their rank position. For example, the spy from the first link may be less trustworthy compared to other spies due to the possible trust bias. Then, in the voting process, we moderate the credibility of the vote from the top rank spies, which is an interesting extension of Algorithm 2. We still need to develop a more sophisticated voting strategy by incorporating continuous probability into the voting procedure to replace the current binary voting method. In the current binary voting strategy, every spy has equal voting power, which implies that every spy is equally important with respect to the final decision. However, in reality, spies could have different trustworthiness. For example, in SpyNB, each spy is already associated with a probability of confidence, which could be used to determine its level of trustworthiness.

We believe that the use of the spying technique in text classification in order to mine preference knowledge is only one of many interesting applications. In general, we could further apply SpyNB in other contexts that need semi-supervised learning in classification. Even in the context of search result personalization, we could further gear the spying technique towards the RSVM directly to mine preferences by voting on the rank order, which is a lightweight approach to the problem. Since the new direction of personalizing a search engine through adapting its ranking function has just emerged, many extensions can be further investigated. As evident in our experiments, the linear ranking function is quite effective for search engine personalization; however, the power of a linear ranking function is still limited compared to more sophisticated ranking functions, e.g., a polynomial ranking function. (Note that the linear function is just a special case of a polynomial function.) We also aim to develop the existing prototype into a full-fledged adaptive search engine. We are considering incremental updates on the ranking function. In other words, whenever the user clicks on the result of a query, the training process is invoked, leading to the optimization of the corresponding ranker. The challenge is that we need to ensure the scalability of the training and optimization processes.

ACKNOWLEDGMENTS

We would like to express our sincere thanks to the editor and the reviewers, who gave very insightful and encouraging comments. This work is supported in part by grants from the Research Grants Council of Hong Kong, Grant Nos. HKUST6165/03E, 616005 and DAG04/05.EG10.

REFERENCES

MSNSearch Search Engine. http://www.msnsearch.com.

Overture Search Engine. http://www.overture.com.

RSVM Code. http://svmlight.joachims.org.

WiseNut Search Engine. http://www.wisenut.com.

Agrawal, R. and Wimmers, E. 2000. A framework for expressing and combining preferences. In *Proc. of the 19th ACM SIGMOD International Conference on Management of Data*. 297–306.

Bartell, B., G.Cottrell, and Belew, R. 1994. Automatic combination of multiple ranked retrieval systemss. In *Proc. of the 17th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 173–181.

Blum, A. and Mitchell, T. 1998. Combining labeled and unlabeled data with co-training. In *Proc. of the 11th Annual Conference on Learning Theory (COLT 98)*. 92–100.

Deng, L., Chai, X., Ng, W., and Lee, D. 2004. Spying out real user preferences for metasearch engine adaptation. In *Proc. of the 6th ACM SIGKDD Workshop on Web Mining and Web Usage Analysis (WebKDD 04)*. Seattle, USA, 71–82.

Goulden, C. 1956. *Methods of Statistics Analysis*, 2nd ed. John Wiley & Sons.

Haveliwala, T. 2002. Topic-sensitive pagerank. In *Proc. of the 11th International World Wide Web Conference (WWW 02)*. 517–526.

Heer, J. and Chi, E. H. 2002. Separating the swarm: categorization methods for user sessions on the web. In *CHI*. 243–250.

Hoffgen, K., Simon, H., and Horn, K. V. 1995. Robust trainability of single neurons. *Journal of Computer and System Sciences 50*, 114–125.

Jeh, G. and Widom, J. 2003. Scaling personalized web search. In *Proc. of the 12th International World Wide Web Conference (WWW 03)*. 271–279.

Joachims, T. 1999. Making large-scale SVM learning practical. In B. Scholkoph et al., editor, *Advances in Kernel Methods – Support Vector Learning*. MIT Press.

Joachims, T. 2002a. Evaluating retrieval performance using clickthrough data. In *Proc. of the SIGIR Workshop on Mathematical/Formal Methods in Information Retrieval*.

Joachims, T. 2002b. Optimizing search engines using clickthrough data. In *Proc. of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 02)*. 133–142.

Joachims, T., Granka, L. A., Pan, B., Hembrooke, H., and Gay, G. 2005. Accurately interpreting clickthrough data as implicit feedback. In *SIGIR*. 154–161.

Ke, Y., Deng, L., Ng, W., and Lee, D. L. 2005. Web dynamics and their ramifications for the development of web search engines. *Computer Networks Journal*.

Kießling, W. 2002. Foundations of preferences in database systems. In *Proc. of the 28th International Conference on Very Large Data Bases (VLDB 02)*. Hong Kong, China, 311–322.

Liu, B., Dai, Y., Li, X., and Lee, W. S. 2003. Building text classifiers using positive and unlabeled examples. In *Proc. of the 3rd IEEE International Conference on Data Mining (ICDM)*.

Liu, B., Lee, W. S., Yu, P., and Li, X. 2002. Partially supervised classification of text documents. In *Proc. of the 19th International Conference on Machine Learning (ICML)*.

Liu, F., Yu, C., and Meng, W. 2002. Personalize web search by mapping user queries to categories. In *Proc. of the 11th ACM International Conference on Information and Knowledge Management (CIKM 02)*. 558–565.

Liu, F., Yu, C., and Meng, W. 2004. Personalized web search for improving retrieval effectiveness. *IEEE Transactions on Knowledge and Data Engineering (TKDE) 16*, 28–40.

McCallum, A. and Nigam, K. 1998. A comparison of event models for naive bayes text classification. In *Proc. of AAAI/ICML-98 Workshop on Learning for Text Categorization*. 41–48.

Mitchell, T. 1997. *Machine Learning*. McGraw Hill, Inc.

Pretschner, A. and Gauch, S. 1999. Ontology based personalized search. In *ICTAI*. 391–398.

Sugiyama, K., Hatano, K., and Yoshikawa, M. 2004. Adaptive web search based on user profile constructed without any effort from users. In *Proc. of the 13th International World Wide Web Conference (WWW 04)*. 675–684.

Tan, Q., Chai, X., Ng, W., and Lee, D. 2004. Applying co-training to clickthrough data for search engine adaptation. In *Proc. of the 9th International Conference on Database Systems for Advanced Applications (DASFAA 04)*. 519–532.

Appendix: The queries used in the online and offline experiments

Computer Science	Social Science	Finance
B tree	Afghan crash	10 day MBA
Convex Hull	China manned space flight	Adobe Photoshop
database	China SARS	air ticket
Finite Automaton	Columbia space shuttle loss	barbie dolls
Gaussian elimination	COMDEX 2003	Canon Photo Printer
Geotools	Fire in Moscow	Database Software
greedy algorithm	Gay civil rights	Digital Image Process
Hamming code	Georgia president resign	Elizabeth Arden
Huffman code	grammy 2003	Farewell My Concubine
image segmentation	HKUST CUHK merge	Finding Nemo
instruction set	HSBC terrorist	flower
Karnaugh maps	crocodile Hong Kong	Garfield
KD-tree	Jackson child abuse	HD600
Kohonen's map	Japan spy satellite	m-audio revolution
latent variable	Miss World 2003	NET MultiSync LCD
LDAP	NBA	Neutrogena
matlab	newegg	OLAP
Multiplexing	Olympics Beijing	OMEGA watch
Oracle	Pakistan India peace talks	Panasonic av100
OSI layer	Palestinians Israeli barrier	Pentax optio
planar graph	Qusay and Uday Hussein	perfume
Polymorphism	Robert Baggio	pocket PC H4100
Quick Sort	SARS report Hong Kong	refrigerator
RAID	sina	sennheiser
sparse matrix	Donald Tsang	sofa
Sunil Arya	Taiwan new vote law	SonyEricsson P900
UPD protocols	Turks bomb synagogues	Tablet PC
vector space model	War in Iraq	Visual Studio
machine learning	WTO	Web cam
XML	Saddam captured	Windows XP

Table X. Queries used in the offline experiment

Computer Science	Social Science	Finance
apriori algorithm	Al Qaeda	American Wedding
AVL tree	Afghanistan Kabul	Battery Pack
bayesian networks	al-Qaeda attack warning	Bruce Almighty
Broadcast disk	Arnold role after election	Canon PowerShot A80
CISI collection	ATLANTA severe flu	Christian Dior
Cosine Similarity	Baghdad blast	digital camera
De Morgan's Theorem	Bush visit Iraq	Discman
Delauney triangulation	California gubernatorial	Flash Memory
Dik Lun Lee	China property right	Fortune magazine
Directed Graph	former Congo dictator	Harry Potter
dynamic programming	China firework bomb	Hello Ketty
eulerian graph	Gay marriage	Hi-Fi
infinite automaton	Gaza blast	Intel CPU
hidden markov model	Georgia opposition bomb	Jewelry pendant
k means clustering	Howard Dean	Lord of ring
metasearch engine	Iran quake	Microsoft Office
mobile wireless protocol	Karbala attacks	New York Times
Overriding	Kuala Lumpur Kidnappers	Nokia 6610
PGPS	Lost electricity America Canada	Norton security
Process control block	Moscow tragedy	Panasonic DVD player
R Tree	Mugabe Commonwealth	American Wedding
radix sort	Libya nuclear	Panasonic plasma TV
SGML	SARS outbread again	Panasonic SD card
singular matrix	Somalia terrorist haven	Shiseido
stoplist download	Song Meiling died	Snoopy
support vector machine	Spanish agents killed Iraq	Sony VAIO V505D
TinyDB	Staten island ferry crash	SQL Sever
TREC collection	Strong quake Philippines	Suisse Programme
UML	Benin plane crash	The Pianist
zipf distribution	Turks bomb synagogues	Tungsten

Table XI. Queries used in the online experiment