Spying Out Real User Preferences for Metasearch Engine Personalization

Lin Deng **Xiaoyong Chai** **Qingzhao Tan** **Wilfred Ng** **Dik Lun Lee**
Department of Computer Science
Hong Kong University of Science and Technology
Email: {ldeng, carnamel, ttqqzz, wilfred, dlee}@cs.ust.hk

Abstract
Most current metasearch engines provide uniform service to users but do not cater for the specific needs of individual users. To address this problem, research has been done on personalizing a metasearch engine. An interesting and practical approach is to optimize its ranking function using clickthrough data. However, it is still challenging to infer accurate user preferences from the clickthrough data. In this paper, we propose a novel learning technique called "Spy Naïve Bayes" (SpyNB) to identify the user preference pairs generated from clickthrough data. We then employ ranking SVM to build a metasearch engine optimizer. To evaluate the effectiveness of SpyNB on ranking quality, we develop a metasearch engine prototype that comprises three underlying search engines: MSNSearch[1], WiseNut[2] and Overture[3] to conduct experimental evaluation. The empirical results show that, compared with the original ranking, SpyNB can significantly improve the average ranks of users' click by 20%, while the performance of the existing methods are not satisfactory.

Key Words: Spy naïve Bayes, search engine personalization, clickthrough, user preferences.

1. Introduction

The information on the World Wide Web is huge and growing rapidly. An effective search engine is an important means for users to find their desired information from billions of Web pages. In particular, metasearch engines are frequently used nowadays, which allow users to access multiple search engines simultaneously with an uniform interface.

Personalization of Web search is to carry out Web information retrieval incorporating a user's interests captured from his search histories [14]. There are two aspects that need to be addressed to personalize a metasearch engine. On the one hand, the underlying search engine components may have different strengths in terms of coverage and speciality and thus different retrieval qualities for different categories of queries. Therefore, there is a *query-specific personalization* problem. On the other hand, users may have diversified preferences on the retrieved Web pages, which produce a *user-specific personalization* problem. A personalized metasearch engine should be able to adapt its ranking function for different categories of queries and different communities of users who share similar interests. Therefore, it is a challenging task to optimize the ranking function of a metasearch engine to cater for users' preferences.

Some previous studies made use of users' explicit relevance feedback to adapt search engine's ranking function [2, 5]. Nevertheless, users are usually unwilling to give explicit feedback because of the manual efforts involved. Explicit feedback data are costly and usually too little to be representative. To overcome this problem, researchers have recently studied the use of clickthrough data, which is a kind of implicit relevance feedback data, to optimize the ranking functions [4, 10] in an automatic manner.

Formally, clickthrough data is represented as a triplet (q, r, c), where q is the input query, r is the list of ranked results presented to the user, $(l_1, \ldots, l_n)$, and c is the set of links that a user has clicked on. Figure 1 illustrates an example of clickthrough data presented in [18], in which the submitted query q is "Biometrics

[1] http://www.msn.com
[2] http://www.wisenut.com
[3] http://www.overture.com

Research". In the figure, three links l_1, l_7 and l_{10} are in bold, which means that they have been clicked on by the user.

Links	Information of web pages in the search results
l_1 clicked	**Biometrics Research Page** biometrics.cse.msu.edu
l_2	National Cancer Institute - Biometric Research linus.nci.nih.gov/ brb
l_3	International Biometric Group www.biometricgroup.com
l_4	Microsoft Research - Vision Technology research.microsoft.com/research/vision
l_5	Forest Biometrics Research Institute www.forestbiometrics.com/Institute.htm
l_6	Signal Processing Research Centre www.sprc.qut.edu.au/research/fingerprint.html
l_7 clicked	**Research: Biometrics** www.nwfusion.com/research/biometrics.html
l_8	Biometrics: Overview biometrics.cse.msu.edu/info.html
l_9	TeKey Research Group www.tekey.com
l_{10} clicked	**Biometrics Research Areas IRL** www.research.ibm.com/irl/projects/biometrics

Figure 1. The clickthrough data for the query "Biometrics Research"

The main advantage of using clickthrough data is that it does not require extra effort by the user, and thus can be obtained at very low cost. However, clickthrough data can be noisy and sometimes ambiguous. As a consequence, it is more difficult to interpret clickthrough data than explicit relevance feedback data.

In essence, there are two steps to optimize the ranking function with respect to users' preferences of a metasearch engine. The first step is to identify user preferences which are usually represented as pairs of clickthrough data items. The second step is to optimize the ranking function which takes into account the preference pairs provided by the first step. There exists an effective algorithm, ranking SVM [10] in the second step, while little research has investigated the first step previously. In this paper, we focus on the first step and reinvestigate the problem: *How do we accurately elicit user preference information from clickthrough data?*

We propose a novel learning technique called *Spy Naïve Bayes* (SpyNB), which analyze the titles, abstracts and URLs of returned links to identify the actual irrelevant links. The SpyNB provides a natural way to discover the real user preference information from implicit feedback data (e.g. clickthrough), for the reason that users actually made their relevance judgements by examining on the titles, abstracts and URLs.

The rest of this paper is organized as follows. In Section 2, we briefly review the related works. In Section 3, we present our SpyNB algorithm to identify user preference information from clickthrough data. In Section 4, we revisit the idea of ranking SVM. In Section 5, the experimental results are reported. Section 6 discusses related issues and Section 7 concludes the paper.

2. Related Work

The work related to the process of learning metasearch engine rankers using clickthrough data falls in two areas. The first area is the study of optimization of a ranking function in search engines. The second area is

the study of analyzing clickthrough data to extract users' preference information. However, most previous literature has investigated the first area, while little research has focus on the second area.

Regarding the work of optimization of a ranking function, Joachims [10] proposes an effective algorithm called ranking SVM to learn the retrieval function of a metasearch engine using user preference pairs as input data. Recently, [18] extends the ranking SVM algorithm to a co-training framework in order to solve the problem of lack of clickthrough data. The co-training algorithm used in [18] is proposed in [3].

Regarding the work of analyzing clickthrough data, to the best of our knowledge, the only method is proposed by Joachims [10, 9], which elicits *partially* relative user preference pairs from clickthrough data. We call this method "*Joachims method*" throughout this paper. Joachims method assumes that the users scan the ranking presented from top to bottom. Therefore, if a user skips link l_i and clicks on link l_j which ranks lower than link l_i $(i < j)$. Joachims method assumes that the user must have observed link l_i and decided not to click on it. Then user preference pairs are elicited as $l_j <_{r'} l_i$ [10, 9], where $<_{r'}$ represents a user's preference order of items of the returned result.

As an example shown in Figure 1, the user did not click on l_2, l_3, l_4, l_5, and l_6, but clicked on l_7. Therefore according to *Joachims method*, l_7 is more relevant to the user than the other five links. In other words, l_7 should rank ahead of those five links in the target ranking. Similarly, l_{10} should rank ahead of l_2, l_3, l_4, l_5, l_6, l_8, and l_9 in the target ranking. Let r' denote the ranking extracted from clickthrough data by *Joachims method*. All preference pairs obtained on this clickthrough are shown in Figure 2.

Preference pairs arising from l_1	Preference pairs arising from l_7	Preference pairs arising from l_{10}
Empty Set	$l_7 <_{r'} l_2$	$l_{10} <_{r'} l_2$
	$l_7 <_{r'} l_3$	$l_{10} <_{r'} l_3$
	$l_7 <_{r'} l_4$	$l_{10} <_{r'} l_4$
	$l_7 <_{r'} l_5$	$l_{10} <_{r'} l_5$
	$l_7 <_{r'} l_6$	$l_{10} <_{r'} l_6$
		$l_{10} <_{r'} l_8$
		$l_{10} <_{r'} l_9$

Figure 2. Preference pairs derived from clickthrough data using Joachims method

3. Learning Preference Pairs from clickthrough Data

We first discuss some inadequacy of Joachims method of learning preference pairs. Then we propose our new interpretation of clickthrough data.

3.1 Inadequacy of Partially Preference Pairs

As shown in Figure 2, Joachims method only captures *partially* relative preference, which means that there are many links incomparable with each other (e.g., l_1 and l_2, l_7 and l_8 are incomparable when paired with respect to $<_{r'}$). Another observation can be made is that Joachims method is apt to penalize high-ranked links. That is, the high-ranked links are more likely to appear in the right hand side of preference pairs (E.g. l_2 and l_3), which means unpreferred by the user, than the left hand side of preference pairs, which means preferred by the user.

We argue to remove some assumptions in Joachims method and reinvestigate the problem of obtaining preference pairs.

First, Joachims' assumptions on obtaining preference pairs are strong. In reality, users may not strictly scan the presented results from top to bottom. Also, if a user skips a link, it may due to other reasons, aside from the skipped link being relatively irrelevant. For example, it may be the reason that the abstract

of skipped links is not informative. Therefore, the preference pairs derived with Joachims method may not be accurate to understand users' preferences.

Second, but more importantly, Joachims method is apt to penalize high-ranked links, as most of the high-ranked links appear on the right hand side of the preference pairs. Our experimental results (Section 5) indicate that, the immoderate penalty on high-ranked links can cause a poor generalization performance. We will elaborate more on this point in Section 6.

3.2 New Clickthrough Data Interpretation

It is obvious that users do not click on links at random, but make an informed choice. Typically, users click on links whose titles, abstracts or URLs are interesting to them. Although clickthrough data is usually noisy, the clicks still convey some user preference information for a large set of clickthrough data. It is therefore reasonable to regard the clicked links as positive samples, which means that they match user preference from a statistical point of view. On the other hand, users are usually disinclined to click on all links that match their information needs. Therefore, it is also reasonable to take all the unclicked links as a set of unlabeled samples, which contains links either matching or not matching users' information needs.

Based on the interpretation of clickthrough data described above, the problem becomes how to identify some *reliable negative* links from the unlabeled set, where *negative* means unpreferred or does not match user's interests. After the underlying reliable negatives are identified, the user preference can be reflected that the user prefers all links in the positive set to those in the negative set. Let P denote the positive set, U denote the unlabeled set and RN denote the reliable negative set, where $RN \subset U$. The user preference pairs can be represented as follows:

$$l_i <_{r'} l_j, \forall l_i \in P, l_j \in RN \tag{1}$$

Equation 1 indicates that all links from the positive set should rank ahead of those from the reliable negative set in the target ranking.

As an example, suppose a user uses a search engine to find some research institutes working on "biometric research" and gets the returned results shown in Figure 1. After manually examining the results, we can see that only the links l_6 and l_9 are not about biometric research institutes. However, the user does not click on all the other links, instead only clicks on links l_1, l_7 and l_{10}. That is because the user may not have enough patience to click all links that match his information needs. Therefore, harshly penalizing high-ranked links may lead to mistakes. If we have an effective approach to identify that links l_6 and l_9 are the actual irrelevant links, the generated user preference pairs will be more accurate.

3.3 Spy Naïve Bayes Technique

How to identify the reliable negative examples from the unlabeled set using only positive and unlabeled data? Recently, *partially supervised classification* [11, 12, 13, 19] provides a novel paradigm that constructs classifiers using positive examples and a large set of unlabeled examples. Finding reliable negative examples is a crucial step of partially supervised classification, and several techniques have been proposed, such as the Spy technique [13], 1-DNF [19], and Rocchio method [11]. These techniques have been witnessed to be effective in the text classification domain. We further incorporate the Spy technique [13] with a novel voting procedure into the naïve Bayes classifier, which we call *Spy Naïve Bayes* (SpyNB) to identify the reliable negative examples.

We first illustrate how the Naïve Bayes [16] is adapted in our context as follows. Let "+" and "–" denote the positive and negative classes, respectively. Let $L = \{l_1, l_2, \cdots, l_N\}$ denote a set of N links (documents) in the search results. Each link l_i can be described as a word vector $(w_1, w_2, \cdots, w_M)$ in the vector space model [1], where the value of w_i indicates the times of word w_i appearing in the titles, abstracts and URLs. Then, a naïve Bayes classifier is built by estimating the prior probabilities ($Pr(+)$ and $Pr(-)$) and likelihoods ($Pr(w_j|+)$ and $Pr(w_j|-)$), as shown in Algorithm 1.

In Algorithm 1, $\delta(+|l_i)$ indicates the class label of link l_i, whose value is 1 if l_i is positive, and 0 otherwise. $Num(w_j, l_i)$ is a function counting the number of keyword w_j appears in link l_i. λ is the smoothing factor, where $\lambda = 1$ is known as the Laplacian smoothing [15], which we use in our experiments.

Algorithm 1 Naïve Bayes Algorithm

Input: L – a set of links $\{l_1, l_2, \cdots, l_N\}$

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

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

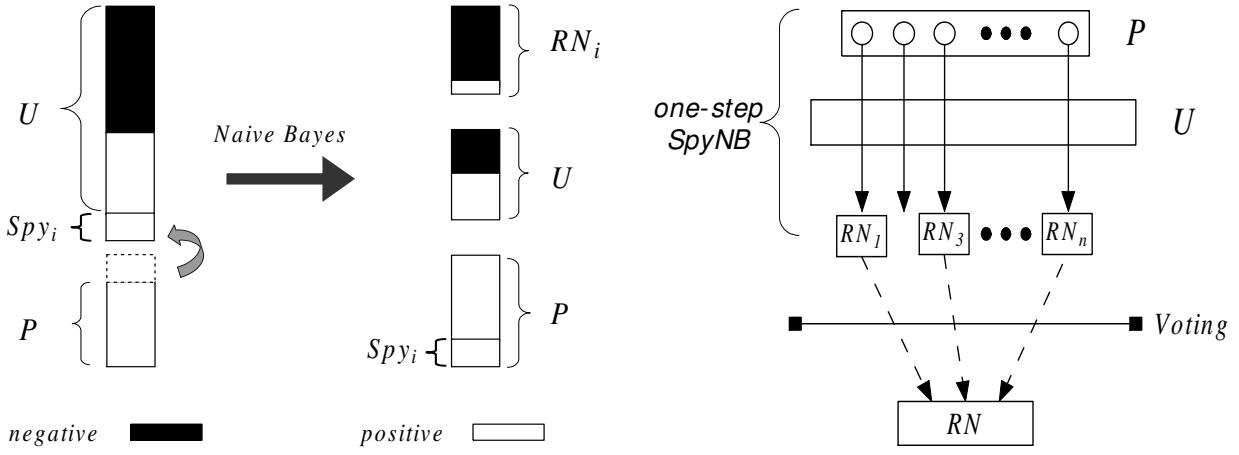
6: **end for**

Output: Prior probabilities – $Pr(+)$ and $Pr(-)$; Likelihoods – $Pr(w_j|+)$ and $Pr(w_j|-)$ $(j = 1, \cdots, M)$

When testing, Naïve Bayes classifies an link l by calculating the posterior probability using 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 $P(+|l)$ is larger than $P(-|l)$; and "–" otherwise.

When training data contain only positive and unlabeled examples, the "Spy" technique can be introduced to learn a naïve Bayes classifier [13]. The procedure is shown in Figure 3(a). First, a set of positive examples S are selected from P and put in U, to act as "spies". Then, the unlabeled examples in U together with S are taken as negative to train naïve Bayes using Algorithm 1. The obtained classifier is then used to assign probabilities $Pr(+|l)$ to each example in $U \cup S$. After that, a threshold T_s is decided by the probabilities assigned to S. An unlabeled example in U is selected as a reliable negative example if its probability is less than T_s, and thus RN is obtained. The examples in S acts as "spies", since they are regarded as positive examples and are put in U pretending as negative examples. During classification, the unknown positive examples in U is assumed to behave similar to the spies (be assigned comparative probabilities). Therefore, the reliable negative examples RN can be identified.

In [13], the Spy technique is used in text classification and 15% of positive examples are randomly selected as spies. The spies are used only once. However, for our metasearch engine optimization problem, the clickthrough data has some distinct characteristics compared with common texts. For instance, the titles and abstracts both are very short texts, and the size of positive set (usually several links per query) is also much smaller than that in text classification domain. As a consequence, the identified RN is not reliable if only a small part of positive examples are used as spies.



(a) One step of SpyNB

(b) Voting procedure of SpyNB

Figure 3. Spy Naïve Bayes Technique

To overcome this problem, we incorporate a novel voting procedure (Figure 3(b)) with "Spy" technique and propose a Spy Naïve Bayes (SpyNB) algorithm. The idea of the voting procedure of SpyNB is as follows. The algorithm runs a n-times iteration, where $n = |P|$ is the number of positive examples. In each iteration, a positive example p_i in P is selected to act as a spy. It is then put in U to train a naïve Bayes classifier NB_i. The probability $Pr(+|p_i)$ assigned to the spy p_i can be used as a threshold T_s to select a candidate reliable negative example set (RN_i). That is, any unlabeled example u_j with a smaller probability of being a positive example than the spy ($Pr(+|u_j) < T_s$) will be selected into RN_i. And consequently, n candidate reliable negative sets: $RN_1, RN_2, \cdots, RN_n$ are identified. Then, a voting procedure is taken to combine all RN_i into the final RN. An unlabeled example is included in the final RN, if and only if, it appears in at least a certain portion (T_v) of RN_i. The advantage of adopting the voting procedure in SpyNB is that the procedure makes full use of all positive examples in P. Also, the procedure makes decisions on RN by taking opinions from all possible spies and thus minimize the influence of random selection of spies.

The SpyNB algorithm is given in Algorithm 2. Steps (2-15) generates n candidates of reliable negative example sets RN_i, and Steps (16-20) combines all RN_i into the final RN, using the voting procedure.

Algorithm 2 Spy Naïve Bayes Algorithm

Input: P – a set of positive examples; U – a set of unlabeled examples; T_v – a voting threhold; T_s – a spy threshold
Procedure:

1: $RN_1 = RN_2 = \cdots = RN_{|P|} = \{\}$ and $RN = \{\}$;
2: **for** each example $p_i \in P$ **do**
3: $P_s = P - \{p_i\}$;
4: $U_s = U \cup \{p_i\}$;
5: Assign each example in P_s the class label 1;
6: Assign each example in U_s the class label -1;
7: Build a naïve Bayes classifier on P_s and U_s;
8: Classify each example in U_s using the NB classifier;
9: Spy threshold $T_s = Pr(+|p_i)$;
10: **for** each $u_j \in U$ **do**
11: **if** $Pr(+|u_j) < T_s$ **then**
12: $RN_i = RN_i \cup \{u_j\}$;
13: **end if**
14: **end for**
15: **end for**
16: **for** each $u_j \in U$ **do**
17: $Votes$ = the number of RN_i such that $u_j \in RN_i$
18: **if** $Votes > T_v \times |P|$ **then**
19: $RN = RN \cup \{u_j\}$;
20: **end if**
21: **end for**

Output: RN – the set of reliable negative examples

The time complexity of the SpyNB algorithm is $O(|P| \times N)$, where $|P|$ is the number of positive examples and N is the number of all examples, including positive and unlabeled. For typical clickthrough data, $|P|$ usually is less than ten, and N is usually less than one hundred. Therefore, the SpyNB algorithm is efficient on clickthrough data in practice.

4. Optimizing Ranking Functions

After user preference pairs are identified with SpyNB, we employ a ranking SVM [10] to optimize the ranking function using the generated preference pairs. We now briefly revisit the basic idea of ranking SVM.

For a training data set, $T = \{(q_1, r'_1), (q_2, r'_2), \ldots, (q_n, r'_n)\}$, where q_i in T is a query and r'_i is the corresponding target ranking, ranking SVM aims at finding a linear ranking function $f(q, d)$, which holds as

many preference pairs in T as possible. $f(q,d)$ is defined as the inner product of a *weight vector* $\overrightarrow{\omega}$ and a *feature vector* of query-document mapping $\phi(q,d)$. $\phi(q,d)$ describes how well a document d of a link in the ranking matches a query q (will be detailed in Section 5.2). $\overrightarrow{\omega}$ gives a weighting of each feature.

Given a weight vector, $\overrightarrow{\omega}$, retrieved links can be ranked by sorting the values: $f(q,d) = \overrightarrow{\omega} \cdot \phi(q,d)$. Then, the problem of finding a ranking function, f, becomes finding a weight vector, $\overrightarrow{\omega}$, that makes the maximum number of the following inequalities hold:

$$\text{For all} \quad (d_i, d_j) \in r'_k, \ \ (1 \leq k \leq n): \qquad \overrightarrow{\omega} \cdot \phi(q_k, d_i) > \overrightarrow{\omega} \cdot \phi(q_k, d_j) \tag{2}$$

where $(d_i, d_j) \in r'_k$ is a document pair corresponding to the preference pair $(l_i <_{r'_k} l_j)$ of q_k, which means d_i should rank higher than d_j in the target ranking of r'_k. Figure 4 illustrates the effect of different weight vectors on ranking three documents, d_1, d_2 and d_3, while the target ranking is $d_1 <_{r^*} d_2 <_{r^*} d_3$. As we can see, $\overrightarrow{\omega_1}$ is better than $\overrightarrow{\omega_2}$: the documents are correctly ranked as (d_1, d_2, d_3) by $\overrightarrow{\omega_1}$, but are ranked as (d_2, d_1, d_3) by $\overrightarrow{\omega_2}$ in which $d_1 < d_2$ does not hold.

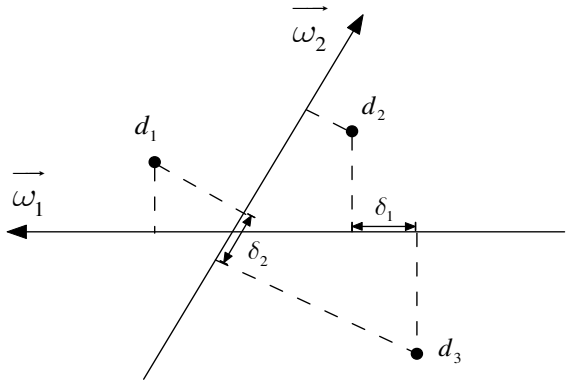


Figure 4. Ranking the documents d_1, d_2, and d_3 with the weight vectors $\overrightarrow{\omega_1}$ and $\overrightarrow{\omega_2}$

However, solving $\overrightarrow{\omega}$ with the constraints in Equation (2) is *NP-hard* [7]. 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:

$$\text{For all} \quad (d_i, d_j) \in r'_k, \ \ (1 \leq k \leq n): \qquad \overrightarrow{\omega} \cdot \phi(q_k, d_i) > \overrightarrow{\omega} \cdot \phi(q_k, d_j) + 1 - \xi_{ijk}, \quad \xi_{ijk} \geq 0 \tag{3}$$

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

$$V(\overrightarrow{\omega}, \xi) = \frac{1}{2} \overrightarrow{\omega} \cdot \overrightarrow{\omega} + C \sum \xi_{ijk}, \tag{4}$$

subject to the constraints given in Equation (3).

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. 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 make all the rankings hold subject to the condition in Equation (3), the one that maximizes the margin δ is preferred. This is because the larger 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 slack variables in target function (4) is the sum of the errors in ranking pairs. Therefore, minimizing this term can be viewed as minimizing the total training errors made. Finally, parameter C is introduced to allow a trade-off between the margin size δ and the total training errors.

As the output, ranking SVM gives a weight vector $\overrightarrow{\omega}$, which can be used to rank future retrieved results by sorting the value: $f(q,d) = \overrightarrow{\omega} \cdot \phi(q,d)$. The ranking SVM algorithm is implemented in a SVM-Light software, which can be downloaded at [8].

5. Experiments

5.1 Experiment Setup

In order to evaluate the effectiveness of our method, we develop a metasearch engine prototype that comprises MSNsearch, WiseNut and Overture. Then we exploit the *unbiased experiment setup* designed in [9] and run the experiments as follows. When a user submits a query to our system, the query is sent to the three search engines. Then, the top 10 links from each search engine are captured and the combined list is produced in a round-robin way as described in [9, 10]. If a result is returned by more than one search engine, we only present it once. The titles, abstracts and URLs of the retrieved results are displayed to users in a uniform style. Therefore, the users do not know that a particular link is from which search engine.

To collect clickthrough data, we ask five graduate students in Computer Science Department, HKUST to test our system. The users are considered to have the same interests as they come from the same community. We gave the users three categories of queries for searching: computer science (CS), news and shopping; and each category contains 30 queries. As thus, the experiment aims to test our ranking function optimizer in a context of *query-specific* personalization. But in general, our method can also work for *user-specific* personalization. We specify the queries and do not let users test their own queries freely for the reason that in a query-specific context, we need to know what category a query exactly belongs to. Overall, 340 clicks are captured for the whole data set. Some statistics of the data set are provided in Figure 5.

Query category	Computer Science	News	Shopping
Number of queries	30	30	30
Number of clicks	123	87	130
Avg. clicks per query	4.1	2.9	4.3
Avg. rank clicked on	5.87	5.6	5.59

Figure 5. Statistics on the data set

5.2 Feature Extraction

Defining a feature vector of the query-document mapping $\phi(q, d)$ is important for the experiments. We now present how the features are extracted in our experiments.

It is known that a metasearch engine does not crawl the Web pages and maintain its own indexes. Therefore, it is hard to represent the retrieved documents as a vector of all words contained in the document, because this information is not available for a metasearch engine. On the other hand, other information such as the ranks in the returned list of underlying search engines can be used to represent documents. Basically, the features used in our experiments are defined according to common intuition about what information is important for learning a ranking function. Substantially, they are the same as those used in [10] and [18].

In our experiments, we totally extract 20 features to define the feature vector of $\phi(q, d)$. We classify all features into three categories, namely, *Rank Features*, *Common Features* and *Similarity Features*.

1. Rank Features (3 numerical features and 12 binary features).

   Let search engine $E \in \{M, W, O\}$ (M stands for MSNsearch, W for WiseNut, and O for Overture) and $T \in \{1, 3, 5, 10\}$. We define numerical features $Rank\_E$ and binary features Top_E_T as follows:

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

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

2. Common Features (2 binary features).

   - *Com_2*: If document d ranks top 10 in at least two search engines, the value is 1, otherwise 0.
   - *Com_3*: If document d ranks top 10 in three search engines, the value is 1, otherwise 0.

3. Similarity Features (1 binary and 2 numerical features).

   - *Sim_U*: The similarity between query and URL is defined as follows:

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

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

The feature vector $\phi(q, d)$ is defined with all the above extracted features as given below:

$$(Rank\_M, Top\_M\_1, \ldots, Rank\_W, \ldots, Rank\_O, \ldots, Top\_O\_10, Com\_2, \ldots, Sim\_U, \ldots, Sim\_A)$$

5.3 Experimental Results

We conduct offline experiments to verify the effectiveness of our SpyNB method on the data set described above. We compare our SpyNB method with the Joachims method [10] described in Section 2. As stated before, Joachims method harshly penalizes high-ranked links, which leads to a poor generalization performance. To verify this point, we simply modify Joachims method to make a "mJoachims" method to generate user preference pairs. The mJoachims method selects some preference pairs with the high-ranked links appearing in the left hand side, to alleviate the penalty on high-ranked links by standard Joachims method. To illustrate the idea of mJoachims method, suppose l_i is a clicked link, l_j is the next clicked link right after l_i (that is, none of clicked links exists between l_i and l_j), and l_k is any skipped link ranks between l_i and l_j, then the preference pairs derived with mJoachims method are those derived with standard Joachims method added with all pairs $l_i <_{r'} l_k$ $(i < k < j)$. As an example, the preference pairs derived with mJoachims method from the clickthrough data in Figure 1, are shown in Figure 6.

Preference pairs arising from l_1	Preference pairs arising from l_7	Preference pairs arising from l_{10}
$l_1 <_{r'} l_2$	$l_7 <_{r'} l_2$	$l_{10} <_{r'} l_2$
$l_1 <_{r'} l_3$	$l_7 <_{r'} l_3$	$l_{10} <_{r'} l_3$
$l_1 <_{r'} l_4$	$l_7 <_{r'} l_4$	$l_{10} <_{r'} l_4$
$l_1 <_{r'} l_5$	$l_7 <_{r'} l_5$	$l_{10} <_{r'} l_5$
$l_1 <_{r'} l_6$	$l_7 <_{r'} l_6$	$l_{10} <_{r'} l_6$
	$l_7 <_{r'} l_8$	$l_{10} <_{r'} l_8$
	$l_7 <_{r'} l_9$	$l_{10} <_{r'} l_9$

Figure 6. Preference pairs derived from clickthrough data using mJoachims method

We conduct comparison experiments on the three methods: SpyNB, Joachims and mJoachims. The procedure is as follows. We first respectively use the three methods to elicit users' preference pairs from clickthrough data on a training set. After that, we employ the ranking SVM algorithm to learn the ranking functions using the elicited preference pairs. Finally, we test the learned ranking functions on a testing set with the criterion of average rank of users' clicks. We divide the training and testing set based on 3-fold cross-validation [6]. In the experiment, the voting threshold T_v in Algorithm 2 is set as 50%.

The evaluation metric we use is the average rank of users' clicks, distinguishing from that in [10], which has some defects as discussed in Section 6. In terms of our metric, it is obvious that with the same number of users' clicks, the lower the average rank of users' clicks, the better the ranking function.

Comparison results are provided in Figure 7, in which the "Original" means the original ranking presented to the user. To show the actual improvement, we compute the relative values of average rank in Figure 8, which is the average rank of users' clicks obtained by a learning method divided by that of the original rank (values smaller than 1 indicate improvement).

	CS	News	Shopping
Original	5.87	5.6	5.59
SpyNB	4.91	4.42	4.65
Joachims	13.62	15.3	15.8
mJoachims	9.8	10.72	12.03

Figure 7. Comparison on average rank of users' clicks of three methods

	CS	News	Shopping
SpyNB	0.836	0.789	0.832
Joachims	2.32	2.73	2.83
mJoachims	1.67	1.91	2.15

Figure 8. Relative improvement of three methods on original ranking

From the results in Figure 7 and Figure 8, we can see that SpyNB can significantly improve the quality of the ranking function in terms of minimizing the average rank of users' clicks. The values of SpyNB in Figure 8 are about 0.8, which means SpyNB can improve (decrease) the average ranks of users' clicks at about 20%. However, we can see that the performance of Joachims method is not satisfactory. Its average ranks of users' clicks (greater than 1) are even worse than the original ranking. The reason is that Joachims method is harshly penalizing high-ranked documents, while most of the user clicked links in the testing set are high ranked, thus it produces a large average rank of users' clicks. In the experiment, mJoachims method outperforms standard Joachims method, which verifies our statement, while mJoachims method is simply adding some pairs to counteract the penalty on high-ranked links of Joachims method.

What do the learned ranking functions look like? As detailed in Section 5.2, the learned ranking function in our experiment is a weight vector comprising 20 components. We list the weight vectors learned on the query categories of "Computer Science", and "Shopping" in Figure 9 and Figure 10 respectively. Intuitively, the features with high absolute weights have large impacts on the resulted ranking. In particular, a higher positive (negative) weight indicates that the links with this feature will be ranked higher (lower) in the combined list. As we can see, the weight vector of the "Computer Science" category and the "Shopping" category are quite distinguishable, which clearly indicates that the underlying search engines have different strengths in terms of topical specialty.

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

Figure 9. Learned weight vector of the "Computer Science" category

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

Figure 10. Learned weight vector of the "Shopping" category

We can also draw some user preference information of the group of users in our experiment from the learned weight vector. Generally speaking, the numerical *Rank Features*: $Rank\_M$, $Rank\_O$ and $Rank\_W$

reflect the relative importance of MSNSearch, Overture and WiseNut respectively. As we can see from Figure 9 and Figure 10, the values of $Rank\_M$ are the largest for both the "Computer Science" and the "Shopping" categories. The value of $Rank\_O$ is small for the "Computer Science" category, but large (almost equal to $Rank\_M$) for the "Shopping" category. Moreover, the values of $Rank\_W$ are relative small for both categories. These observations indicate that MSNSearch are strong in all the queries in both categories, Overture is particularly good at shopping queries, and WiseNut does not have outstanding performance in any query category. These observations are consistent with the recent survey on search engines [17], which claims that MSNSearch is considered as one of the best general search engines in the world, Overture is an advertising search engine, and WiseNut is a growing search engine which still needs to perfect itself.

6. Discussion

In this section, we further discuss two interesting problems related to this work. The first problem is why harshly penalizing high-ranked links as shown in Joachims method can lead to a *over generalization* of a ranking function, which is undesirable for ranking quality. The second problem is why the evaluation metric "prediction error" used in [10] is not applicable in our offline experiments.

As for the first problem, we need to explain what "over generalization" means. Let us use the clickthrough of the query "biometric research" in Table 1 as an example, in which links l_1, l_7 and l_{10} are clicked. Let us also assume this clickthrough is the only example available in the training set. Intuitively, the ranking function derived with Joachims method will then make the links l_7 and l_{10} rise up, but the links l_2,...,l_6 fall down in a testing process. However, the training and testing are conducted on a set of more than one clickthroughs and most of them are not of "biometric research" queries. Thus, the training and testing processes are overly "generalized" by the links l_7 and l_{10}, since these links may not be clicked links in many testing clickthroughs. In reality, most of users' clicks in testing set are high-ranked links, since the users typically are disinclined to see low-ranked results. Therefore, Joachims method is apt to penalize high-ranked links, and make high-ranked links unnecessarily fall down when testing. Such harsh penalization on high-ranked links makes the Joachims method have a poor ranking performance.

As for the second problem, the evaluation metric used in [10] is the "prediction error" parameter, which is defined as the percentage of preference pairs generated in the first step that are not fulfilled after the links in a testing example are ranked by a learned ranking function. We exemplify the computational process of the clickthrough data shown in Table 2 as follows. For this clickthrough, there are totally 12 preference pairs identified by Joachims method (in Table 1). Suppose that after this clickthrough is ranked by a learned ranking function, 9 out of 12 pairs are fulfilled, then the "prediction error" value is computed as: $\frac{12-9}{12} = 25\%$.

This "prediction error" metric can be used to evaluate the ranking SVM algorithm in [10], by assuming the preference pairs derived with Joachims method as correct rules. However, the preference pairs identified by SpyNB and Joachims method are different, and thus the "prediction error" parameters of them are incomparable. On the other hand, a ranking function with a low "prediction error" can still produce a large value of average rank of users' clicks. In our experiments, the ranking function derived with Joachims method can achieve a quite good "prediction error" at about 17% on the "computer science" category. However it produces the average rank of users' clicks more than a double of the original value. Thus, we adopt the *average rank of users' clicks* parameter in our experiments, which we consider a better evaluation metric than the "prediction error" used in [10].

7. Conclusions and Future Work

This paper presents an effective approach to personalize metasearch engines using clickthrough data. Our approach consists of two learning steps. In the first step, we propose a novel SpyNB method to identify user preference pairs from clickthrough data. After that, we employ ranking SVM to optimize the ranking function using the preference pairs identified in the first step. The experimental results demonstrate that our method can significantly improve the ranking quality in terms of the average rank of users' clicks compared with the original ranking and the existing methods.

As a future work, we are going to enhance our metasearch engine system and to make it available on the Web. We also plan to conduct online experiments to evaluate our method.

References

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-wesley-Longman, Harlow, UK, 1999.

[2] B. Bartell, G.Cottrell, and R. Belew. Automatic combination of multiple ranked retrieval systemss. In *Proc. of the 17th ACM SIGIR Conference*, pages 173–181, 1994.

[3] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proc. of the 11th COLT Conference*, pages 92–100, 1998.

[4] J. Boyan, D. Freitag, and T. Joachims. A machine learning architecture for optimizing web search engines. In *Proc. of AAAI workshop on Internet-Based Information System*, 1996.

[5] W. Cohen, R. Shapire, and Y. Singer. Learning to order things. *Journal of Artifical Intelligence Research*, 10:243–270, 1999.

[6] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Willey and Sons, Inc., New York, 2nd edition, 2001.

[7] K. Hoffgen, H. Simon, and K. Van Horn. Robust trainability of single neurons. *Journal of Computer and System Sciences*, 50:114–125, 1995.

[8] T. Joachims. Making large-scale SVM learning practical. In B. Scholkoph et al., editor, *Advances in Kernel Methods – Support Vector Learning*. MIT Press, 1999. http://svmlight.joachims.org/.

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

[10] T. Joachims. Optimizing search engines using clickthrough data. In *Proc. of the 8th ACM SIGKDD Conference*, pages 133–142, 2002.

[11] X. Li and B. Liu. Learning to classify text using positive and unlabeled data. In *Proc. of 8th International Joint Conference on Artificial Intelligence*, 2003.

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

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

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

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

[16] T. Mitchell. *Machine Learning*. McGraw Hill, Inc., 1997.

[17] D. Sullivan. Major search engines and directories. Electronic Articles in Search Engine Watch, April 2004. http://searchenginewatch.com.

[18] Q. Tan, X. Chai, W. Ng, and D.L. Lee. Applying co-training to clickthrough data for search engine adaptation. In *Proc. of the 9th DASFAA conference*, pages 519–532, 2004.

[19] H. Yu, J. Han, and K. Chang. PEBL: Positive example based learning for web page classification using svm. In *Proc. of the 8th ACM SIGKDD Conference*, 2002.