Spying Out Accurate User Preferences for Search Engine Adaptation

Lin Deng, Wilfred Ng, Xiaoyong Chai, and Dik-Lun Lee

Department of Computer Science
Hong Kong University of Science and Technology
{ldeng, wilfred, carnamel, dlee}@cs.ust.hk

Abstract. Most existing search engines employ static ranking algorithms that do not adapt to the specific needs of users. Recently, some researchers have studied the use of clickthrough data to adapt a search engine's ranking function. Clickthrough data indicate for each query the results that are clicked by users. As a kind of implicit relevance feedback information, clickthrough data can easily be collected by a search engine. However, clickthrough data is sparse and incomplete, thus, it is a challenge to discover accurate user preferences from it. In this paper, we propose a novel algorithm called "Spy Naïve Bayes" (SpyNB) to identify user preferences generated from clickthrough data. First, we treat the result items clicked by the users as *sure positive* examples and those not clicked by the users as unlabelled data. Then, we plant the sure positive examples (the spies) into the unlabelled set of result items and apply a naïve Bayes classification to generate the *reliable negative* examples. These positive and negative examples allow us to discover more accurate user's preferences. Finally, we employ the SpyNB algorithm with a ranking SVM optimizer to build an adaptive metasearch engine. Our experimental results show that, compared with the original ranking, SpyNB can significantly improve the average ranks of users' click by 20%.

1 Introduction

The information on the Web is huge and growing rapidly. An effective search engine is an important means for users to find the desired information from billions of Web pages. Besides standalone search engines, metasearch engines are also very useful because they allow users to access multiple search engines simultaneously with a uniform interface.

Adapting a search engine to cater for specific users and queries is an important research problem and has many applications. In general, there are two aspects of search engine adaptation that need to be addressed. The first aspect is *query specific* adaptation; that is, how to return the best results for a query from the underlying search engines that have different coverage and focuses. The second aspect is *user specific* adaptation that aims to meet the diversified preferences of different users in the search results. A well-adapted metasearch engine should be able to optimize its ranking function for different query categories and

different user communities. The challenging task is how to adapt the ranking function of a metasearch engine to cater for different users' preferences.

Some previous studies employed users' explicit relevance feedback to adapt the search engine's ranking function [1, 2]. However, users are usually unwilling to give explicit feedback because of the manual efforts involved, making the feedback data too few 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 [3–5] in an automatic manner.

Formally, clickthrough data are represented as a triplet (q, r, c), where q is the input query, r is the ranked result links $(l_1, \ldots, l_n)$, and c is the set of links that a user has clicked on. Figure 1 illustrates an example from clickthrough data of the query "apple" and three links l_1, l_7 and l_{10} are in bold, indicating that they have been clicked by the user. The main advantage of using clickthrough data is that it does not require extra effort from the user, and thus can be obtained at a very low cost. However, clickthrough data are ambiguous when used as a sign of user preferences. As a consequence, it is more difficult to interpret clickthrough data and discover user preferences than explicit relevance feedback data.

In essence of search engine adaptation using clickthrough data, there are two steps. The first step is to identify user preferences (i.e., the user prefers one result over another). The second step is to optimize the ranking function based on the preferences obtained in the first step. There exists an effective algorithm called ranking SVM [4] for the second step, but little research has been done for the first step. In this paper, we focus on the accurate elicitation of user preferences from clickthrough data. In particular, we propose a novel learning technique called *Spy Naïve Bayes* (SpyNB), which analyzes the titles, abstracts and URLs of the returned links to identify any actual irrelevant links. We show that SpyNB is an effective way to discover accurate user preferences from clickthrough data by incorporating SpyNB (for the first step) with a ranking SVM (for the second step) to construct an adaptive metasearch engine ranker. Notice that SpyNB can be used to adapt the ranking function of a standalone search engine. However, metasearch is chosen in this paper, since it does not only serve as an important search tool but allows us to focus on the adaptive ranking of the results without considering crawling and indexing, which are not the goal of our paper. In addition, metasearch allows us to choose underlying search engines with different strengths, coverages and focuses, thus giving us a new dimension to observe the effectiveness of SpyNB.

Finally, we develop a metasearch engine prototype that comprises MSNSearch, WiseNut and Overture for conducting experimental performance evaluations. The empirical results show that SpyNB algorithm can accurately elicit user preferences from clickthrough data and thus improve the ranking quality of a metasearch engine. Importantly, the ranking quality produced with SpyNB is shown to be significantly better than that of Joachims algorithm and the original rankings from the underlying search engines.

Links	The list of search results with titles, abstracts and URLs of webpages
$\mathbf{l_1}$ **(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*
l_4	www.apple-history.com A brief history of the company that changed the computing world . . . *http://www.apple-history.com/*
l_5	MacCentral: Apple Macintosh News Steve Jobs unveils Apple mini stores. . . . *http://www.macworld.com/news/*
l_6	Adams County Nursery, apple trees One of the most widely planted apple cultivars worldwide. *http://www.acnursery.com/apples.htm*
$\mathbf{l_7}$ **(clicked)**	**Apple .Mac Welcome** **. . . member specials throughout the year. See . . .** ***http://www.mac.com/***
l_8	AppleInsider . . . Apple seeds Mac OS X Server 10.3.6 build $7R20$. *http://www.appleinsider.com/*
l_9	ROSE APPLE Fruit Facts The rose apple is too large to make a suitable container plant. . . . *http://www.crfg.org/pubs/ff/roseapple.html*
$\mathbf{l_{10}}$ **(clicked)**	**Apple - Support** **Support for most Apple products provided by Apple Computer** ***http://www.info.apple.com/***

Fig. 1. A clickthrough for the query "apple". Links in bold are clicked by the user.

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 preferences from clickthrough data. In Section 4, we revisit the idea of a ranking SVM. In Section 5, the experimental results related to the effectiveness of our SpyNB algorithm are reported. Section 6 concludes the paper.

2 Related Work

Related work on search engine adaptation using clickthrough data falls into two subareas. The first one is the analysis of clickthrough data to identify user's preferences. The second one is the study of the optimization of a search engines' ranking function using the identified preferences. For ranking function optimization,

ranking SVM[4] is an effective algorithm, which can learn an optimized ranking function using user preferences as input. Recently, an RSCF algorithm [5] has been proposed to extend ranking SVM to a co-training framework in order to tackle the lack of clickthrough data for training.

For clickthrough data analysis, a simple algorithm was proposed by Joachims [4], which elicits preference pairs from clickthrough data. We call this method "*Joachims algorithm*" throughout this paper. Joachims algorithm assumed that the user scans the ranked results strictly 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 algorithm assumed that the user must have observed link l_i and decided not to click on it. Then preference pairs are elicited as $l_j <_{r'} l_i$, where $<_{r'}$ represents the target ranking of search results.

For example, in the clickthrough of the "apple" query shown in Figure 1, the user does not click on l_2, l_3, l_4, l_5, and l_6, but clicks on l_7. Therefore according to Joachims algorithm, 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. All preferences obtained using Joachims algorithm 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	$l_7 <_{r'} l_2$	$l_{10} <_{r'} l_2$
	$l_7 <_{r'} l_3$	$l_{10} <_{r'} l_3$
	$l_7 <_{r'} l_4$	$l_{10} <_{r'} l_4$
	$l_7 <_{r'} l_5$	$l_{10} <_{r'} l_5$
	$l_7 <_{r'} l_6$	$l_{10} <_{r'} l_6$
		$l_{10} <_{r'} l_8$
		$l_{10} <_{r'} l_9$

Fig. 2. Preferences derived from the clickthrough of Figure 1 using Joachims algorithm

3 Learning Preferences from Clickthrough Data

We first discuss some inadequacies of Joachims algorithm. We then introduce a new interpretation of clickthrough data, and based on that the SpyNB algorithm for learning preferences from clickthrough data.

3.1 Inadequacy of Joachims algorithm

As depicted in Section 2, Joachims algorithm is simple and efficient. However, we argue that the assumption made by Joachims algorithm of how users scan search results is too strong in reality, since users' behaviors are diverse. Therefore, there

could be a problem in that Joachims algorithm assumes that the user scans search results *strictly* from top to bottom, as in reality a user may leap over several results. In short, the up-to-down scanning may not in reality be *strict*.

Moreover, we notice that Joachims algorithm is unfair to the high-ranking links, which means that the high-ranking links (e.g. l_1, l_2) are more likely to be "penalized" than the low-ranking links (e.g. l_9, l_{10}). Consider the preference example shown in Figure 2. Link l_1 and l_{10} are both clicked links; however l_1 appears on the left hand side of preference pairs (meaning it should be ranked high in target ranking) much *less* than l_{10}. (l_1, 0 times; l_{10}, 7 times.) On the other hand, link l_2 and l_9 are both unclicked links; however, l_2 appears on the right hand side of preference pairs (means it should be ranked low in target ranking) *more* than l_9. (l_2, twice; l_9, 1 times.) Therefore, the high-ranking links (e.g. l_1, l_2) are more likely to be ranked low after learning. We note the phenomenon where Joachims algorithm is apt to penalize the high-ranking links.

3.2 New Clickthrough Interpretation

In order to address the above problems, we propose to interpret the clickthrough data in a new manner. We note that the user typically clicks on the links whose titles, abstracts or URLs are interesting to them. Therefore, we assume the clicked links are liked by the user. Moreover, users in general are unlikely to click all the links that match his interests. For example, after a user has obtained the desired information, he stops scanning the results. Thus, we further assume that the unclicked links contain both the links that the user likes and dislikes. Finally, we assume that the disliked links are the links that are most different in content to the clicked links.

Based on the new interpretation, we label the clicked links as *positive* and the unclicked links as *unlabeled* samples. Then the problem of discovering user's preferences becomes how to identify the reliable *negative* samples from the unlabeled set, where *negative* indicates that the link does not match user's interests. After the reliable negatives are identified, the user preference can be reflected in the way 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 pairwise preferences can be represented as:

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

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

3.3 Spy Naïve Bayes

The problem now can be formulated as how to identify the reliable negative examples from an unlabeled set using only positive and unlabeled data. Recently, *partially supervised classification* [6–9] provides a novel paradigm for constructing classifiers using positive examples and a large set of unlabeled examples.

Finding reliable negative examples can be solved by partially supervised classification techniques, such as Spy technique [8], 1-DNF [9], and the Rocchio method [6]. In particular, we incorporate the spy technique with naïve Bayes to design a *Spy Naïve Bayes* (SpyNB) algorithm for identifying the reliable negative examples. We choose the spy technique, because it has been shown to be effective for common text classification [8]. However, clickthrough data have some unique characteristics compared to common texts. For instance, the titles and abstracts are both very short texts, and the size of positive set (the number of clicked links) is also very small. Consequently, the identified RN is not reliable if only a small portion of positive examples are used as spies. Thus we further employ a voting procedure to strengthen SpyNB. In this section, we elaborate on the SpyNB algorithm in detail.

We first illustrate how the Naïve Bayes [10] (NB for short) is adapted in clickthrough analysis 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, $W = (w_1, w_2, \cdots, w_M)$, in the *vector space model* [11], where we count the occurrences of w_i appearing in the titles, abstracts and URLs. Then, a NB classifier is built by estimating the prior probabilities ($Pr(+)$ and $Pr(-)$), and likelihood ($Pr(w_j|+)$ and $Pr(w_j|-)$), as shown in Algorithm 1.

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

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

6: **end for**

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 keywords w_j appearing in link l_i. λ is the smoothing factor, where $\lambda = 1$ is known as Laplacian smoothing [12], which we use in our experiments.

When testing, NB classifies a 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 the training data contain only positive and unlabeled examples, spy technique is used to learn an NB classifier [8]. The idea behind spy technique is shown in Figure 3. 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 regarded as negative to train a NB classifier 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 on 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 act as "spies", since they are regarded as positive examples and are put into U pretending to be negative examples. During classification, the unknown positive examples in U are assumed to have comparative probabilities with the spies. Therefore, the reliable negative examples RN can be identified.

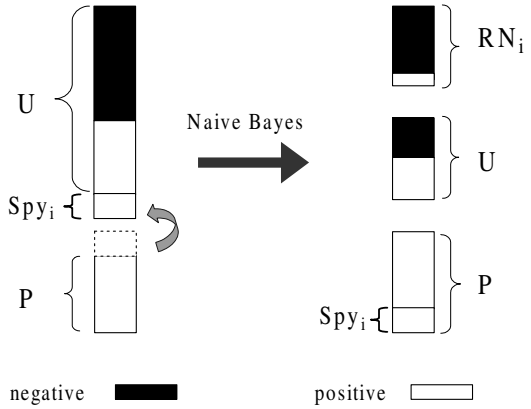


Fig. 3. Spy technique of SpyNB

Due to the unique characteristics of clickthrough data, we further employ a voting procedure (Figure 4) to make Spy Naïve Bayes more robust. The idea of voting procedure is as follows. The algorithm runs an n-time 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 into U to train an NB 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 for a 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 ($Pr(+|u_j) < T_s$) is selected into RN_i. 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 minimizes the influence of a random selection of spies.

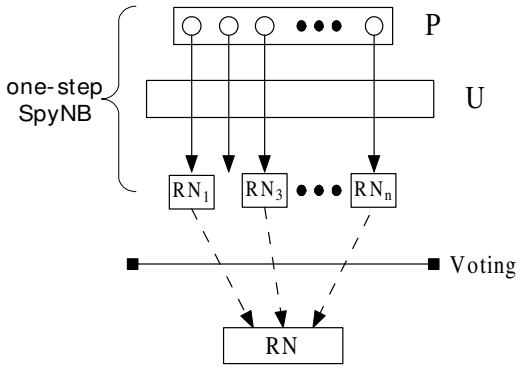


Fig. 4. Voting procedure of SpyNB

The SpyNB algorithm is given in Algorithm 2. Steps 2 to 15 use the Spy technique to generate n candidates of reliable negative example sets RN_i. Steps 16 to 21 combine all RN_i into the final RN using the voting procedure.

To analysis the time complexity of SpyNB algorithm, 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 time scan of all links, thus the time complexity of training is $O(N)$. The predication of naïve Bayes costs $O(|U|)$ time, where $|U| < N$. Thus the steps 2 to 15 of SpyNB algorithm cost $O(|P| \times (N+|U|)) = O(|P| \times N)$ time. With similar analysis, the time complexity of steps 16 to 21 of SpyNB algorithm is $O(|P| \times |U|)$, which is smaller than $O(|P| \times N)$.

Thus overall, the time complexity of the SpyNB algorithm is $O(|P| \times N)$. We know that the time complexity of Joachims algorithm is $O(N)$. It seems that SpyNB algorithm is not as efficient as Joachims algorithm. However, we note that in most realistic cases, $|P|$ (the number of links clicked by the user) is usually very small, and actually can be considered as having a constant bound. For example, the empirical clickthrough data reported in [5] has merely an average of 2.94 clicks per query. Therefore, without losing the generality, we can assume no user

Algorithm 2 Spy Naïve Bayes (SpyNB) Algorithm

Input:

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

Output:

RN – the set of reliable negative examples

Procedure:

$RN_1 = RN_2 = \cdots = RN_{|P|} = \{\}$ and $RN = \{\}$;
for each example $p_i \in P$ **do**
  $P_s = P - \{p_i\}$;
  $U_s = U \cup \{p_i\}$;
  Assign each example in P_s the class label 1;
  Assign each example in U_s the class label -1;
  Build a NB classifier on P_s and U_s;
  Classify each example in U_s using the NB classifier;
  Spy threshold $T_s = Pr(+|p_i)$;
  for each $u_j \in U$ **do**
    if $Pr(+|u_j) < T_s$ **then**
      $RN_i = RN_i \cup \{u_j\}$;
    end if
  end for
end for
for each $u_j \in U$ **do**
  $Votes$ = the number of RN_i such that $u_j \in RN_i$
  if $Votes > T_v \times |P|$ **then**
    $RN = RN \cup \{u_j\}$;
  end if
end for

clicks more than 10 (or any large enough integer) links for a query. Then, $|P| < 10$ and the time complexity of SpyNB algorithm becomes $O(10 \times N) = O(N)$.

In short, although the SpyNB algorithm is more sophisticated than Joachims algorithm, due to the characteristics of clickthrough data, the *practical* complexity of SpyNB is still at the same level as Joachims algorithm, according to the previous analysis.

4 Optimizing Ranking Functions

After preferences are identified by SpyNB, we employ a ranking SVM [4] to optimize the ranking function using the identified preferences. We now briefly revisit the basic idea of ranking SVM as follows.

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 preferences 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:

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

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 5 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.

However, solving $\overrightarrow{\omega}$ with the constraints in Equation (2) is *NP-hard* [13]. 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:

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

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

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

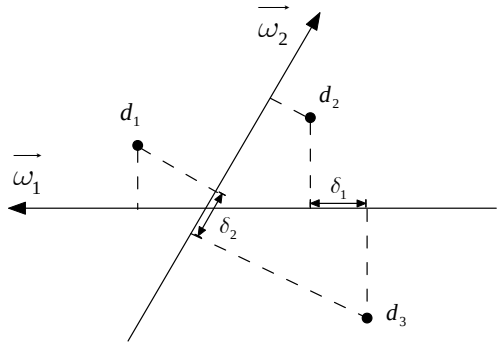


Fig. 5. Ranking the documents d_1, d_2, and d_3 with the weight vectors $\overrightarrow{\omega_1}$ and $\overrightarrow{\omega_2}$

subject to the constraints given in Equation (3).

The idea of solving the above optimization problem is: let δ be the distance between the two closest projected documents along a weight vector. In Figure 5, 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 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 from [14].

5 Experimental Evaluations

We conduct extensive experiments to evaluate our method. Section 5.1 describes how we set up the experiment. Section 5.2 explains the specification of the implemented ranking function. Section 5.3 introduces a baseline method mJoachims algorithm and Section 5.4 reports the experimental results.

5.1 Experiment Setup

In order to evaluate the effectiveness of our method, we develop a metasearch engine that comprises three search engines: MSNSearch[1], WiseNut[2] and Over-

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

ture[3]. At the time of this writing, MSNSearch is regarded as one of the best major search engines in the world; WiseNut is a new and growing search engine; and Overture is an advertising search engine which ranks results based on the prices paid by the sponsors. We choose the three search engines that have different strengths in terms of retrieval quality and focus, as we can test our methods in a *query-specific* adaptation context.

Our metasearch engine works as follows. When a query is submitted to the system, top 100 links from each search engine are retrieved. Then the combined list presented to the user is produced in a round-robin manner [15] to ensure that there is no bias towards any source. 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 in a uniform style. Therefore, the users do not know which search engine a particular link is from.

To collect clickthrough data, we asked five graduate students in our department to test the system. The users are considered to share 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. This setting aims to test the methods in a *query-specific* context. However, in essential, the result can also apply to *user-specific* context. Figure 6 shows some statistics of the clickthrough we collected.

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

Fig. 6. Statistics of experiment data

5.2 Linear Ranking Function

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

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

[3] http://www.overture.com

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 of document d as follows:

$$Rank\_E = \begin{cases} \frac{11-X}{10} & \text{if } d \text{ ranks } X \text{ in } E,\ X <= 10 \\ 0 & \text{otherwise.} \end{cases}$$

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

2. **Common Features** (2 binary features).

$$Com\_2 = \begin{cases} 1 & \text{if } d \text{ ranks top 10 in two search engines;} \\ 0 & \text{otherwise.} \end{cases}$$

$$Com\_3 = \begin{cases} 1 & \text{if } d \text{ ranks top 10 in three search engines;} \\ 0 & \text{otherwise.} \end{cases}$$

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

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

$$Sim\_T = \text{Cosine similarity between query and title.}$$

$$Sim\_A = \text{Cosine similarity between query and abstract.}$$

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

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

The corresponding weight vector $\overrightarrow{\omega}$ contains 20 weight values, each of which reflects the importance of a feature in Equation (5). There are other ways to define $\phi(q, d)$ and $\overrightarrow{\omega}$. Our definition only reflects the intuition about what we think are important for a metasearch engine to rank search results and still easy for implementation.

5.3 mJoachims Algorithm

As pointed out in Section 3.1, Joachims algorithm unfairly penalizes the high-ranking links, which in practice may lead to problems. To verify this point, we modify a bit the way of Joachims algorithm generating preference pairs, and call it "mJoachims algorithm". The mJoachims algorithm adds some preferences to standard Joachims algorithm with the high-ranking links appearing in the left hand side, for alleviating the penalty. In particular, 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 preferences derived with mJoachims algorithm are those derived with standard Joachims algorithm added with the pairs of $l_i <_{r'} l_k$ ($i < k < j$). Figure 7 shows the preference pairs derived using mJoachims algorithm from the clickthrough of Figure 1. The difference between Joachims and mJaochims algorithm can be seen by comparing Figure 2 and Figure 7.

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

Fig. 7. Preferences derived using mJoachims algorithm

5.4 Experimental Results

The experiment consists of three parts. We first compare the effectiveness of SpyNB with Joachims and mJoachims algorithm on ranking quality. Secondly, we analyze the effect of training data size on the performance of algorithms. Finally, we make some interesting observation on the learned ranking functions.

Evaluation on Ranking Quality In order to evaluate SpyNB algorithm on ranking quality, we incorporate SpyNB, Joachims and mJoachims algorithms with ranking SVM, and obtain 3 learned ranking functions. Particularly, we set the voting threshold of SpyNB (T_v in Algorithm 2) as 50% just by random. Then we evaluate the ranking functions by using them to *rerank* the original clickthrough, and see if the ranking quality can be improved.

We measure ranking quality in terms of *the average rank of users' clicks*, denoted as Ψ. Intuitively, a good ranking function should rank the user desired links high. Thus, the smaller the value of Ψ, the better the ranking quality. To show the actual improvement, we define a metric *relative average rank of users' clicks*, denoted as Ψ^r, as the Ψ derived from a personalized ranking function divided by the Ψ of the original search result. $\Psi^r < 1$ indicates that an actual improvement is achieved.

The results are shown in Figure 8. First, the values of Ψ^r of SpyNB are all about $0.8 < 1$, which means the ranking function derived with SpyNB can improve the ranking quality by about 20% for all 3 categories. This result indicates that SpyNB algorithm can effectively discover user's preferences from clickthrough data.

Moreover, we find that Joachims algorithm and mJoachims algorithm fail to achieve actual improvement after reranking the original search results, since their Ψ^r values are greater than 1. We explain this finding as because their strong assumptions do not hold on our empirical clickthrough data. Thus the preferences identified by the existing algorithms are incorrect. Particularly, mJoachims algorithm is relatively better than Joachims algorithm, which can be interpreted that Joachims algorithm is apt to penalize high-ranking links, while mJoachims algorithm alleviates this penalty. Finally, we can conclude that the preferences

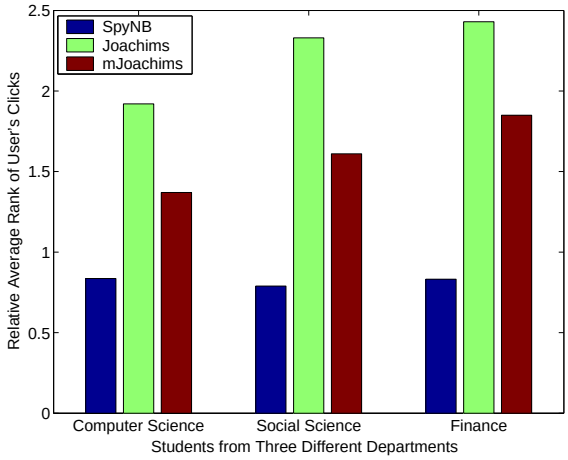


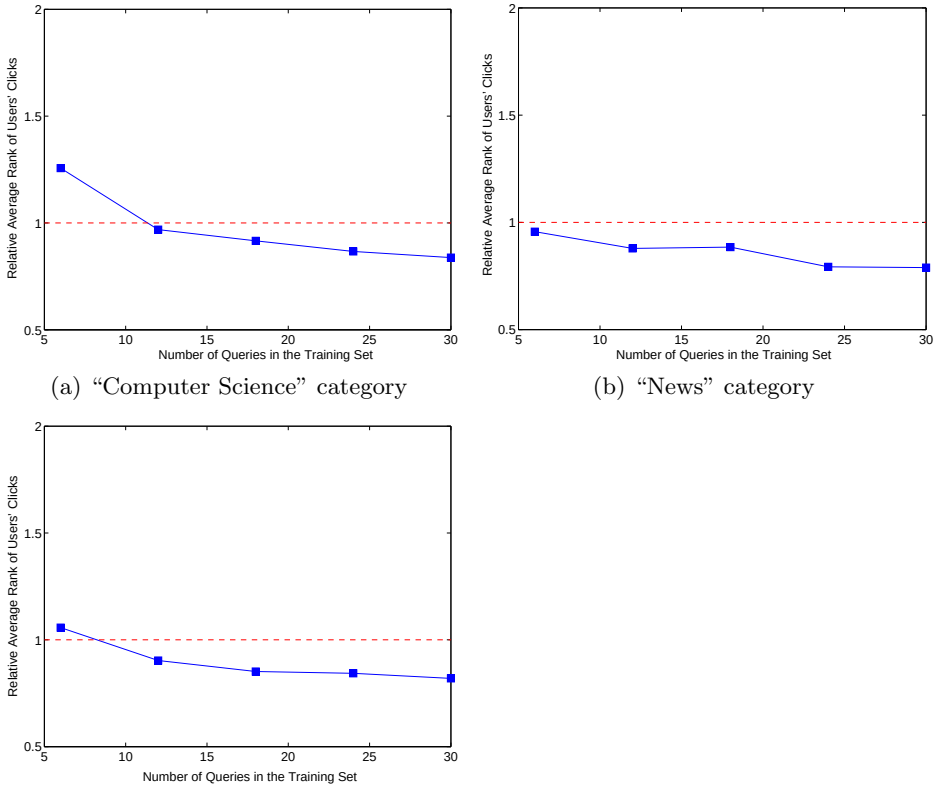
Fig. 8. Comparison on Relative Average Rank of Users' Clicks of three methods

discovered by SpyNB algorithm are much more accurate than those by Joachims and mJoachims algorithms.

Effect of Varying Data Size In order to study the impact of data set size on the ranking function optimizer, we randomly select n queries to evaluate our SpyNB algorithm and the Joachims algorithm, where n is set to 6, 12, 18, 24 and 30. The experimental settings are the same as those described in Section 5.4. We also compute the Ψ^r parameter and present the results in Figure 9.

From Figure 9, we can see that when the data size is small, the performance of SpyNB is not satisfactory (i.e. $\Psi^r > 1$). The reason is that when the data size is too small, the training data is not representative enough for learning an effective ranking function. When the data size is growing, the performance of SpyNB is gradually improved, and when the training data szie increase to 30, the Ψ^r value decreases to around 0.8, which means that SpyNB can achieve about 20% improvement compared with the original ranking. Moreover, we note that the performance curves at point of 30 become quite flat, so we suppose the best performance of SpyNB will converge at some level a bit samller than 0.8. On the other hand, this result also indicates that the least number of clickthrough data for training SpyNB is quite small: just 30 training clickthrough can train an effective SpyNB ranking function optimizer.

Learned Weights of Ranking Functions 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 10 and Figure 11 respectively.



(a) "Computer Science" category (b) "News" category

(c) "Shopping" category

Fig. 9. Relative Average Rank Ψ^r of SpyNB and Joachims algorithm on varying data size

Intuitively, the features with high absolute weights have a large impact on the resulted ranking. In particular, a higher positive (negative) weight indicates that the links with this feature would 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.

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 10 and Figure 11, 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

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

Fig. 10. 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

Fig. 11. Learned weight vector of the "Shopping" category

the queries in both categories, Overture is particularly good at shopping queries, and WiseNut does not perform outstandingly in any query category. These conclusions are consistent with the common knowledge that MSNSearch is 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.

As another interesting observation, we can see that the similarity between query and title seems to be more important than the similarity between query and abstract, for the reason that the values of Sim_T are larger than those of Sim_A in Figure 10 and Figure 11. This observation can be explained as follows: the abstracts sometimes are not very informative and the titles usually have larger influences on users' relevance judgement than the abstracts. In short, analysis of the learned weight vector can be useful to understand the users' preferences and behaviors.

6 Conclusions

In this paper, we first identify some problems of an existing algorithm for discovering user's preferences from clickthrough data. We then introduce a new clickthrough interpretation and propose a novel SpyNB algorithm for discovering preferences based on analyzing the texts (titles, abstracts) of clickthrough data. Furthermore, we present an approach to adapting a search engine's ranking function using SpyNB algorithm plus a ranking SVM optimizer.

To evaluate our methods, we conducted controlled experiments, particularly in a query-specific adaptation context. The experimental results demonstrated that our method significantly improved the ranking quality in terms of the average rank of users' clicks by 20% compared with the original ranking and even more when compared with existing Joachims algorithm.

There are several directions we are going to study in the future. First, we would like to test a more sophisticated Spy Naïve Bayes technique by extending current black-and-white (0 or 1) voting procedure to incorporate continuous

probability into vote values. Moreover, we plan to conduct online interactive experiments to further evaluate our method and also evaluate our method in a user-specific adaptation context.

References

1. Bartell, B., G.Cottrell, Belew, R.: Automatic combination of multiple ranked retrieval systemss. In: Proc. of the 17th ACM SIGIR Conference. (1994) 173–181
2. Cohen, W., Shapire, R., Singer, Y.: Learning to order things. Journal of Artifical Intelligence Research **10** (1999) 243–270
3. Boyan, J., Freitag, D., Joachims, T.: A machine learning architecture for optimizing web search engines. In: Proc. of AAAI workshop on Internet-Based Information System. (1996)
4. Joachims, T.: Optimizing search engines using clickthrough data. In: Proc. of the 8th ACM SIGKDD Conference. (2002) 133–142
5. Tan, Q., Chai, X., Ng, W., Lee, D.: Applying co-training to clickthrough data for search engine adaptation. In: Proc. of the 9th DASFAA conference. (2004) 519–532
6. Li, X., Liu, B.: Learning to classify text using positive and unlabeled data. In: Proc. of 8th International Joint Conference on Artificial Intelligence. (2003)
7. Liu, B., Dai, Y., Li, X., Lee, W.S.: Building text classifiers using positive and unlabeled examples. In: Proc. of the 3rd International Conference on Data Mining. (2003)
8. Liu, B., Lee, W.S., Yu, P., Li, X.: Partially supervised classification of text documents. In: Proc. of the 19th International Conference on Machine Learning. (2002)
9. Yu, H., Han, J., Chang, K.: PEBL: Positive example based learning for web page classification using svm. In: Proc. of the 8th ACM SIGKDD Conference. (2002)
10. Mitchell, T.: Machine Learning. McGraw Hill, Inc. (1997)
11. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-wesley-Longman, Harlow, UK (1999)
12. McCallum, A., Nigam, K.: A comparison of event models for naive bayes text classification. In: Proc. of AAAI/ICML-98 Workshop on Learning for Text Categorization. (1998) 41–48
13. Hoffgen, K., Simon, H., Horn, K.V.: Robust trainability of single neurons. Journal of Computer and System Sciences **50** (1995) 114–125
14. Joachims, T.: 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/.
15. Joachims, T.: Evaluating retrieval performance using clickthrough data. In: Proc. of the SIGIR Workshop on Mathematical/Formal Methods in Information Retrieval. (2002)