

# Search Engines that Learn from Implicit Feedback

Thorsten Joachims and Filip Radlinski, Cornell University

**Search-engine logs provide a wealth of information that machine-learning techniques can harness to improve search quality. With proper interpretations that avoid inherent biases, a search engine can use training data extracted from the logs to automatically tailor ranking functions to a particular user group or collection.**

Each time a user formulates a query or clicks on a search result, easily observable feedback is provided to the search engine. Unlike surveys or other types of explicit feedback, this *implicit feedback* is essentially free, reflects the search engine's natural use, and is specific to a particular user and collection. A smart search engine could use this implicit feedback to learn personalized ranking functions—for example, recognizing that the query “SVM” from users at computer science departments most likely refers to the machine-learning method, but for other users typically refers to ServiceMaster's ticker symbol. With a growing and heterogeneous user population, such personalization is crucial for helping search advance beyond a one-ranking-fits-all approach.<sup>1</sup>

Similarly, a search engine could use implicit feedback to adapt to a specific document collection. In this way, an off-the-shelf search-engine product could learn about the specific collection it's deployed on—for example, learning that employees who search their company intranet for “travel reimbursement” are looking for an expense-report form even if the form doesn't contain the word “reimbursement.” Sequences of *query reformulations* could provide the feedback for this learning task. Specifically, if a significant fraction of employees searching for “travel reimbursement” reformulate the query, and eventually click on the expense-report form, the search engine could learn to include the form in the results for the initial query.<sup>2</sup>

Most large Internet search engines now record queries and clicks. But while it seems intuitive that implicit feed-

back can provide the information for personalization and domain adaptation, it isn't clear how a search engine can operationalize this information. Clearly, implicit feedback is noisy and biased, making simple learning strategies doomed to failure.

We show how, through proper interpretation and experiment design, implicit feedback can provide cheap and accurate training data in the form of pairwise preferences. We provide a machine-learning algorithm that can use these preferences, and demonstrate how to integrate everything in an operational search engine that learns.

## INTERPRETING IMPLICIT FEEDBACK

Consider the example search shown in Figure 1. The user issued the query “Jaguar” and received a ranked list of documents in return. What does the user clicking on the first, third, and fifth link tell us about the user's preferences, the individual documents in the ranking, and the query's overall success?

### User behavior

To answer these questions, we need to understand how users interact with a search engine (click) and how this relates to their preferences. For example, how significant is it that the user clicked on the top-ranked document? Does this tell us that it was relevant to the user's query?

**Viewing results.** Unfortunately, a click doesn't necessarily indicate that a result was relevant, since the way search engines present results heavily biases a user's

behavior. To understand this bias, we must understand the user's decision process.

First, which results did the user look at before clicking? Figure 2 shows the percentage of queries for which users look at the search result at a particular rank before making the first click. We collected the data with an eye-tracking device in a controlled user study in which we could tell whether and when a user read a particular result.<sup>3</sup>

The graph shows that for all results below the third rank, users didn't even look at the result for more than half of the queries. So, on many queries even an excellent result at position 5 would go unnoticed (and hence unclicked). Overall, most users tend to evaluate only a few results before clicking, and they are much more likely to observe higher-ranked results.<sup>3</sup>

**Influence of rank.** Even once a result is read, its rank still influences the user's decision to click on it. The blue bars in Figure 3 show the percentage of queries for which the user clicked on a result at a particular rank. Not surprisingly, users most frequently clicked the top-ranked result (about 40 percent of the time), with the frequency of clicks decreasing along with the rank. To some extent, the eye-tracking results explain this, since users can't click on results they haven't viewed. Furthermore, the top-ranked result could simply be the most relevant result for most queries. Unfortunately, this isn't the full story.

The red bars in Figure 3 show the frequency of clicks after—unknown to the user—we swapped the top two results. In this swapped condition, the second result gained the top position in the presented ranking and received vastly more clicks than the first result demoted to second rank. It appears that the top position lends credibility to a result, strongly influencing user behavior beyond the information contained in the abstract. Note that the eye-tracking data in Figure 2 shows that ranks one and two are viewed almost equally often, so not having viewed the second-ranked result can't explain this effect.

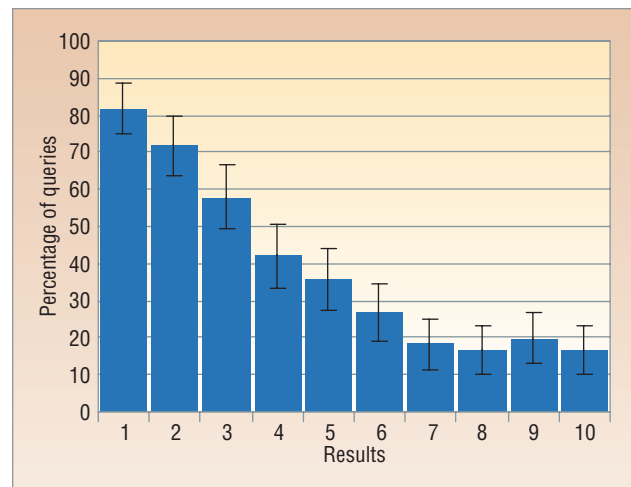
**Presentation bias.** More generally, we found that the way the search engine presents results to the user has a strong influence on how users act.<sup>4</sup> We call the combination of these factors the *presentation bias*. If users are so heavily biased, how could we possibly derive useful preference information from their actions? The following two strategies can help extract meaningful feedback from clicks in spite of the presentation bias and aid in reliably inferring preferences.

### Absolute versus relative feedback

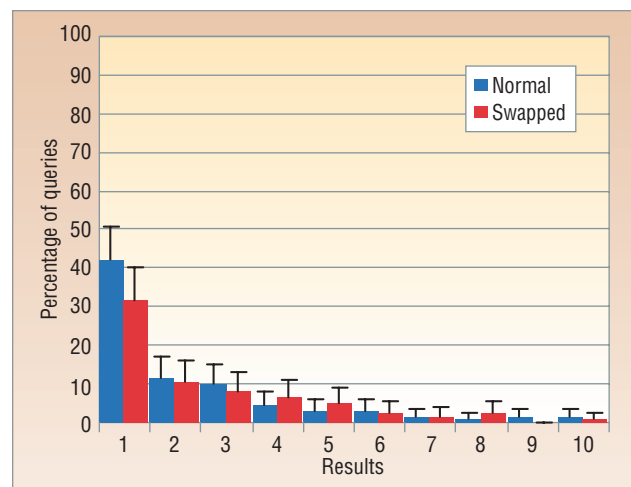
What can we reliably infer from the user's clicks? Due to the presentation bias, it's not safe to conclude that a click indicates relevance of the clicked result. In fact, we find that users click frequently even if we severely

- \*1. The Belize Zoo; <http://belizezoo.org>
2. Jaguar—The British Metal Band; <http://jaguar-online.com>
- \*3. Save the Jaguar; <http://savethejaguar.com>
4. Jaguar UK—Jaguar Cars; <http://jaguar.co.uk>
- \*5. Jaguar—Wikipedia; <http://en.wikipedia.org/wiki/Jaguar>
6. Schrödinger (Jaguar quantum chemistry package); <http://www.schrodinger.com>
7. Apple—Mac OS X Leopard; <http://apple.com/macosx>

**Figure 1. Ranking for “Jaguar” query. The results a user clicks on are marked with an asterisk.**




**Figure 2. Rank and viewership. Percentage of queries where a user viewed the search result presented at a particular rank.**



**Figure 3. Swapped results. Percentage of queries where a user clicked the result presented at a given rank, both in the normal and swapped conditions.**

degrade the quality of the search results (for example, by presenting the top 10 results in reverse order).<sup>3</sup>

**Relative preference.** More informative than what users clicked on is what they didn't click on. For instance, in the example in Figure 1, the user decided



not to click on the second result. Since we found in our eye-tracking study that users tend to read the results from top to bottom, we can assume that the user saw the second result when clicking on result three. This means that the user decided between clicking on the second and third results, and we can interpret the click as a relative preference (“the user prefers the third result over the second result for this query”).

This preference opposes the presentation bias of clicking on higher-ranked links, indicating that the user made a deliberate choice not to click on the higher-ranked result. We can extract similar relative preferences from the user’s decision to click on the fifth result in our example, namely that users prefer the fifth result over the second and fourth results.

The general insight here is that we must evaluate user actions in comparison to the alternatives that the user observed before making a decision (for example, the top  $k$  results when clicking on result  $k$ ), and relative to external biases (for example, only counting preferences that go against the presented ordering). This naturally leads to feedback in the form of pairwise relative preferences like “A is better than B.” In contrast, if we took a click to be an absolute statement like “A is good,” we’d face the difficult task of explicitly correcting for the biases.

**Pairwise preferences.** In a controlled user study, we found that pairwise relative preferences extracted from clicks are quite accurate.<sup>3</sup> About 80 percent of the pairwise preferences agreed with expert labeled data, where we asked human judges to rank the results a query returned by relevance to the query.

This is particularly promising, since two human judges only agree with each other about 86 percent of the time. It means that the preferences extracted from clicks aren’t much less accurate than manually labeled data. However, we can collect them at essentially no cost and in much larger quantities. Furthermore, the preferences from clicks directly reflect the actual users’ preferences, instead of the judges’ guesses at the users’ preferences.

### Interactive experimentation

So far, we’ve assumed that the search engine passively collects feedback from log files. By passively, we mean that the search engine selects the results to present to users without regard for the training data that it might collect from user clicks. However, the search engine has complete control over which results to present and how to present them, so it can run interactive experiments. In contrast to passively observed data, active experiments can detect causal relationships, eliminate the effect of presentation bias, and optimize the value of the implicit feedback that’s collected.<sup>4,5</sup>

**Paired blind experiment.** To illustrate the power of online experiments, consider the simple problem of learning whether a particular user prefers Google or Yahoo! rankings. We formulate this inference problem as a paired blind experiment.<sup>6</sup> Whenever the user types in a query, we retrieve the rankings for both Google and Yahoo!. Instead of showing either of these rankings separately, we combine the rankings into a single ranking that we then present to the user.

Specifically, the Google and Yahoo! rankings are interleaved into a combined ranking so a user reading from top to bottom will have seen the same number of top links from Google and Yahoo! (plus or minus one) at any point in time. Figure 4 illustrates this interleaving technique. It’s easy to show that such an interleaved ranking always exists, even when the two rankings share results.<sup>6</sup> Finally, we make sure that the user can’t tell which results came from Google and which from Yahoo! and that we present both with equally informative abstracts.

We can now observe how the user clicks on the combined ranking. Due to interleaving, a user without preference for Yahoo! or Google will have equal probability of clicking on a link that came from the top of the Yahoo! ranking or the top of the Google ranking. This holds independently of whatever influence the presented rank has on the user’s clicking behavior or however many results the user might consider. If we see that the user clicks significantly more frequently on results from one of the search engines, we can conclude that the user prefers the results from that search engine in this direct comparison.

For the example in Figure 4, we can assume that the user viewed results 1 to 5, since there’s a click on result 5. This means the user saw the top three results from ranking A as well as from ranking B. The user decided to not click on two of the results from B, but did click on all results from A, which indicates that the user prefers A.

**Optimizing data quality.** We can also use interactive experiments to improve the value of the feedback that’s received. The eye-tracking study showed that we can expect clicks only for the top few results, and that the search engine will probably receive almost no feedback about any result ranked below 100. But since the search engine controls the ranking that’s presented, it could mix things up.

While not presenting the current “best-guess” ranking for the query might reduce retrieval quality in the short run, improved training data through exploration can make the search engine learn faster and improve more in the long run. An active learning algorithm for this problem maintains a model of uncertainty about the

Improved training data  
through exploration can  
make the search engine  
learn faster  
and improve more  
in the long run.

relevance of each document for a query, which then allows the search engine to determine which documents would benefit most from receiving feedback.<sup>5</sup> We can then insert these selected documents into the top of the presented ranking, where they will likely receive feedback.

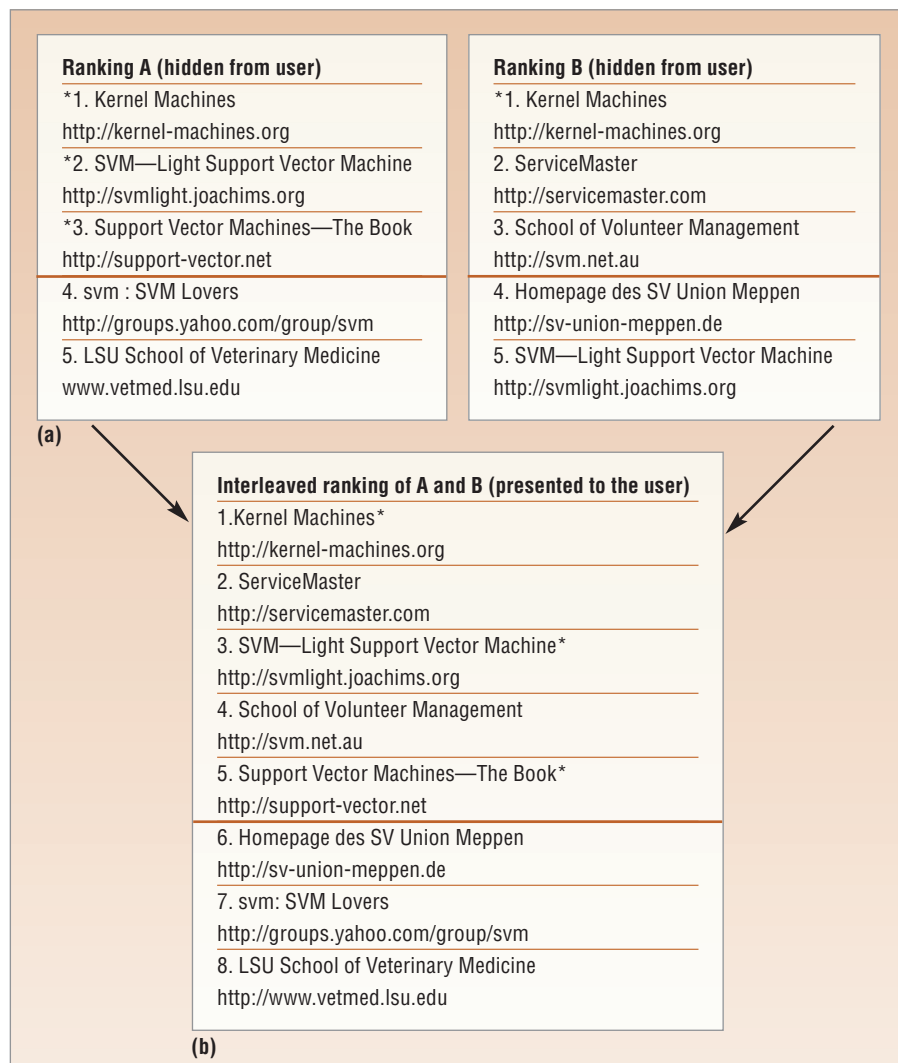
### Beyond counting clicks

So far, the only implicit feedback we've considered is whether the user clicked on each search result. But there's much beyond clicks that a search engine can easily observe, as Diane Kelly and Jaime Teevan note in their review of existing studies.<sup>7</sup> Search engines can use reading times, for example, to further differentiate clicks.<sup>8</sup> A click on a result that's shortly followed by another click probably means that the user quickly realized that the first result wasn't relevant.

**Abandonment.** An interesting indicator of user dissatisfaction is "abandonment," describing the user's decision to not click on any of the results. Abandonment is always a possible action for users, and thus is in line with our relative feedback model. We just need to include "reformulate query" (and "give up") as possible alternatives for user actions. Again, the decision to not click on any results indicates that abandoning the results was the most promising option. Noticing which queries users abandoned is particularly informative when the user immediately follows the abandoned query with another one.

**Query chains.** In one of our studies of Web-search behavior, we found that on average users issued 2.2 queries per search session.<sup>3</sup> Such a sequence of queries, which we call a *query chain*, often involves users adding or removing query terms, or reformulating the query as a whole. Query chains are a good resource for learning how users formulate their information need, since later queries often resolve ambiguities of earlier queries in the chain.<sup>9</sup>

For example, in one of our studies we frequently observed that users searching the Cornell Library Web pages ran the query "oed" followed by the query



**Figure 4. Blind test for user preference.** (a) Original rankings, and (b) the interleaved ranking presented to the user. Clicks (marked with \*) in the interleaved ranking provide unbiased feedback on which ranking the user prefers.

"Oxford English Dictionary." After running the second query, the users often clicked on a particular result that the first query didn't retrieve.<sup>2</sup>

The later actions in this query chain (the reformulation followed by the click) could explain what the user initially intended with the query "oed." In particular, we can infer the preference that, given the query "oed," the user would have liked to see the clicked result returned for the query "Oxford English Dictionary." If we frequently see the same query chain (or more precisely, the resulting preference statement), a search engine can learn to associate pages with queries even if they don't contain any of the query words.

### LEARNING FROM PAIRWISE PREFERENCES

User actions are best interpreted as a choice among available and observed options, leading to relative preference statements like "for query  $q$ , user  $u$  prefers  $d_a$



over  $d_b$ .” Most machine-learning algorithms, however, expect absolute training data—for example, “ $d_a$  is relevant,” “ $d_a$  isn’t relevant,” or “ $d_a$  scores 4 on a 5-point relevance scale.”

## Ranking SVM

How can we use pairwise preferences as training data in machine-learning algorithms to learn an improved ranking? One option is to translate the learning problem into a binary classification problem.<sup>10</sup> Each pairwise preference would create two examples for a binary classification problem, namely a positive example  $(q, u, d_a, d_b)$  and a negative example  $(q, u, d_b, d_a)$ . However, assembling the binary predictions of the learned rule at query time is an NP-hard problem, meaning it’s likely too slow to use in a large-scale search engine.

**Utility function.** We’ll therefore follow a different route and use a method that requires only a single sort operation when ranking results for a new query. Instead of learning a pairwise classifier, we directly learn a function  $h(q, u, d)$  that assigns a real-valued utility score to each document  $d$  for a given query  $q$  and user  $u$ . Once the algorithm learns a particular function  $h$ , for any new query  $q'$  the search engine simply sorts the documents by decreasing utility.

We address the problem of learning a utility function  $h$  from a given set of pairwise preference statements in the context of support vector machines (SVMs).<sup>11</sup> Our ranking SVM<sup>6</sup> extends ordinal regression SVMs<sup>12</sup> to multiquery utility functions. The basic idea is that whenever we have a preference statement “for query  $q$ , user  $u$  prefers  $d_a$  over  $d_b$ ,” we interpret this as a statement about the respective utility values—namely that for user  $u$  and query  $q$  the utility of  $d_a$  is higher than the utility of  $d_b$ . Formally, we can interpret this as a constraint on the utility function  $h(q, u, d)$  that we want to learn:  $h(q, u, d_a) > h(q, u, d_b)$ .

Given a space of utility functions  $H$ , each pairwise preference statement potentially narrows down the subset of utility functions consistent with the user preferences. In particular, if our utility function is linear in the parameters  $\mathbf{w}$  for a given feature vector  $\Phi(q, u, d)$  describing the match between  $q, u$ , and  $d$ , we can write  $h(q, u, d) = \mathbf{w} \cdot \Phi(q, u, d)$ .

Finding the function  $h$  (the parameter vector  $\mathbf{w}$ ) that’s consistent with all training preferences  $P = \{(q^1, u^1, d_a^1, d_b^1), \dots, (q^n, u^n, d_a^n, d_b^n)\}$  is simply the solution of a system of linear constraints. However, it’s probably too much to ask for a perfectly consistent utility function. Due to noise inherent in click data, the linear system is probably inconsistent.

**Training method.** Ranking SVMs aim to find a parameter vector  $\mathbf{w}$  that fulfills most preferences (has low training error), while regularizing the solution with the squared norm of the weight vector to avoid overfitting. Specifically, a ranking SVM computes the solution to

the following convex quadratic optimization problem:

$$\text{minimize: } V(\mathbf{w}, \xi) = \frac{1}{2} \mathbf{w} \cdot \mathbf{w} + C \frac{1}{n} \sum_{i=1}^n \xi_i$$

$$\text{subject to: } \mathbf{w} \cdot \Phi(q^1, u^1, d_a^1) = \mathbf{w} \cdot \Phi(q^1, u^1, d_b^1) + 1 - \xi_1$$

$$\vdots$$

$$\mathbf{w} \cdot \Phi(q^n, u^n, d_a^n) = \mathbf{w} \cdot \Phi(q^n, u^n, d_b^n) + 1 - \xi_n$$

$$\forall i: \xi_i \geq 0$$

We can solve this type of optimization problem in time linear in  $n$  for a fixed precision ([http://svmlight.joachims.org/svm\\_perf.html](http://svmlight.joachims.org/svm_perf.html)), meaning it’s practical to solve on a desktop computer given millions of preference statements. Note that each unsatisfied constraint incurs a penalty  $\xi_i$ , so that the term  $\sum \xi_i$  in the objective is an upper bound on the number of violated training preferences. The parameter  $C$  controls overfitting like in a binary classification SVM.

## A LEARNING METASEARCH ENGINE

To see if the implicit feedback interpretations and ranking SVM could actually learn an improved ranking function, we implemented a metasearch engine called Striver.<sup>6</sup> When a user issued a query, Striver forwarded the query to Google, MSN Search (now called Windows Live), Excite, AltaVista, and HotBot. We analyzed the results these search engines returned and extracted the top 100 documents. The union of all these results composed the candidate set  $K$ . Striver then ranked the documents in  $K$  according to its learned utility function  $h^*$  and presented them to the user. For each document, the system displayed the title of the page along with its URL and recorded user clicks on the results.

## Striver experiment

To learn a retrieval function using a ranking SVM, it was necessary to design a suitable feature mapping  $\Phi(q, u, d)$  that described the match between a query  $q$  and a document  $d$  for user  $u$ . Figure 5 shows the features used in the experiment.

To see whether the learned retrieval function improved retrieval, we made the Striver search engine available to about 20 researchers and students in the University of Dortmund’s artificial intelligence unit, headed by Katharina Morik. We asked group members to use Striver just like any other Web search engine.

After the system collected 260 training queries with at least one click, we extracted pairwise preferences and trained the ranking SVM on these queries. Striver then used the learned function for ranking the candidate set  $K$ . During an approximately two-week evaluation, we compared the learned retrieval function against Google,

MSN Search, and a nonlearning meta-search engine using the interleaving experiment. In all three comparisons, the users significantly preferred the learned ranking over the three baseline rankings,<sup>6</sup> showing that the learned function improved retrieval.

**Learned weights.** But what does the learned function look like? Since the ranking SVM learns a linear function, we can analyze the function by studying the learned weights. Table 1 displays the features that received the most positive and most negative weights. Roughly speaking, a high-positive (or -negative) weight indicates that documents with these features should be higher (or lower) in the ranking.

The weights in Table 1 reflect the group of users in an interesting and plausible way. Since many queries were for scientific material, it was natural that URLs from the domain “citeseer” (and the alias “nec”) would receive positive weight. Note also the high-positive weight for “.de,” the country code domain name for Germany. The most influential weights are for the cosine match between query and abstract, whether the URL is in the top 10 from Google, and for the cosine match between query and the words in the URL. A document receives large negative weights if no search engine ranks it number 1, if it’s not in the top 10 of any search engine (note that the second implies the first), and if the URL is long. Overall, these weights nicely matched our intuition about these German computer scientists’ preferences.

## Osmot experiment

To show that in addition to personalizing to a group of users, implicit feedback also can adapt the retrieval function to a particular document collection, we implemented Osmot ([www.cs.cornell.edu/~filip/osmot](http://www.cs.cornell.edu/~filip/osmot)), an engine that searches the Cornell University Library Web pages. This time we used subjects who were unaware that the search engine was learning from their actions. The search engine was installed on the Cornell Library homepage, and we recorded queries and clicks over several months.<sup>2</sup>

We then trained a ranking SVM on the inferred preferences from within individual queries and across query chains. Again, the search engine’s performance improved significantly with learning, showing that the search engine could adapt its ranking function to this collection.

Most interestingly, the preferences from query chains allowed the search engine to learn associations between queries and particular documents, even if the documents

1. Rank in other search engines (38 features total):	
<b>rank_X</b> :	100 minus rank in $X \in \{\text{Google, MSN Search, AltaVista, HotBot, Excite}\}$ divided by 100 (minimum 0)
<b>top1_X</b> :	ranked #1 in $X \in \{\text{Google, MSN Search, AltaVista, HotBot, Excite}\}$ (binary {0, 1})
<b>top10_X</b> :	ranked in top 10 in $X \in \{\text{Google, MSN Search, AltaVista, HotBot, Excite}\}$ (binary {0, 1})
<b>top50_X</b> :	ranked in top 50 in $X \in \{\text{Google, MSN Search, AltaVista, HotBot, Excite}\}$ (binary {0, 1})
<b>top1count_X</b> :	ranked #1 in $X$ of the five search engines
<b>top10count_X</b> :	ranked in top 10 in $X$ of the five search engines
<b>top50count_X</b> :	ranked in top 50 in $X$ of the five search engines
2. Query/content match (three features total):	
<b>query_url_cosine</b> :	cosine between URL-words and query (range [0, 1])
<b>query_abstract_cosine</b> :	cosine between title-words and query (range [0, 1])
<b>domain_name_in_query</b> :	query contains domain-name from URL (binary {0, 1})
3. Popularity attributes (~20,000 features total):	
<b>url_length</b> :	length of URL in characters divided by 30
<b>country_X</b> :	country code $X$ of URL (binary attribute {0, 1} for each country code)
<b>domain_X</b> :	domain $X$ of URL (binary attribute {0, 1} for each domain name)
<b>abstract_contains_home</b> :	word “home” appears in URL or title (binary attribute {0,1})
<b>url_contains_tilde</b> :	URL contains “~” (binary attribute {0,1})
<b>url_X</b> :	URL $X$ as an atom (binary attribute {0,1})

Figure 5. Striver metasearch engine features. Striver examined rank in other search engines, query/content matches, and popularity attributes.

Table 1. Features with largest and smallest weights as learned by the ranking SVM.

Weight	Feature
0.60	query_abstract_cosine
0.48	top10_google
0.24	query_url_cosine
0.24	top1count_1
0.24	top10_msnsearch
0.22	host_citeseer
0.21	domain_nec
0.19	top10_count_3
0.17	top1_google
0.17	country_de
...	
-0.13	domain_tu-bs
-0.15	country
-0.16	top50count_4
-0.17	url_length
-0.32	top10count_0
-0.38	top1count_0

didn’t contain the query words. Many of the learned associations reflected common acronyms and mis-

spellings unique to this document corpus and user population. For example, the search engine learned to associate the acronym “oed” with the gateway to dictionaries and encyclopedias, and the misspelled name “lexus” with the Lexis-Nexis library resource. These findings demonstrated the usefulness of pairwise preferences derived from query reformulations.

**T**aken together, these two experiments show how using implicit feedback and machine learning can produce highly specialized search engines. While biases make implicit feedback data difficult to interpret, techniques are available for avoiding these biases, and the resulting pairwise preference statements can be used for effective learning. However, much remains to be done, ranging from addressing privacy issues and the effect of new forms of spam, to the design of interactive experiments and active learning methods. ■

### Acknowledgments

This work was supported by NSF Career Award No. 0237381, a Microsoft PhD student fellowship, and a gift from Google.

### References

1. J. Teevan, S.T. Dumais, and E. Horvitz, “Characterizing the Value of Personalizing Search,” to be published in *Proc. ACM SIGIR Conf. Research and Development in Information Retrieval* (SIGIR 07), ACM Press, 2007.
2. F. Radlinski and T. Joachims, “Query Chains: Learning to Rank from Implicit Feedback,” *Proc. ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining* (KDD 05), ACM Press, 2005, pp. 239-248.
3. T. Joachims et al., “Evaluating the Accuracy of Implicit Feedback from Clicks and Query Reformulations in Web Search,” *ACM Trans. Information Systems*, vol. 25, no. 2, article 7, 2007.
4. F. Radlinski and T. Joachims, “Minimally Invasive Randomization for Collecting Unbiased Preferences from Clickthrough Logs,” *Proc. Nat’l Conf. Am. Assoc. for Artificial Intelligence* (AAAI 06), AAAI, 2006, pp. 1406-1412.
5. F. Radlinski and T. Joachims, “Active Exploration for Learning Rankings from Clickthrough Data,” to be published in *Proc. ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining* (KDD 07), ACM Press, 2007.
6. T. Joachims, “Optimizing Search Engines Using Clickthrough Data,” *Proc. ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining* (KDD 02), ACM Press, 2002, pp. 132-142.
7. D. Kelly and J. Teevan, “Implicit Feedback for Inferring User Preference: A Bibliography,” *ACM SIGIR Forum*, vol. 37, no. 2, 2003, pp. 18-28.
8. E. Agichtein, E. Brill, and S. Dumais, “Improving Web Search Ranking by Incorporating User Behavior,” *Proc. ACM SIGIR Conf. Research and Development in Information Retrieval* (SIGIR 06), ACM Press, 2006, pp. 19-26.
9. G. Furnas, “Experience with an Adaptive Indexing Scheme,” *Proc. ACM SIGCHI Conf. Human Factors in Computing Systems* (CHI 85), ACM Press, 1985, pp. 131-135.
10. W.W. Cohen, R.E. Shapire, and Y. Singer, “Learning to Order Things,” *J. Artificial Intelligence Research*, vol. 10, AI Access Foundation, Jan.-June 1999, pp. 243-270.
11. V. Vapnik, *Statistical Learning Theory*, John Wiley & Sons, 1998.
12. R. Herbrich, T. Graepel, and K. Obermayer, “Large-Margin Rank Boundaries for Ordinal Regression,” P. Bartlett et al., eds., *Advances in Large-Margin Classifiers*, MIT Press, 2000, pp. 115-132.

*Thorsten Joachims is an associate professor in the Department of Computer Science at Cornell University. His research interests focus on machine learning, especially with applications in information access. He received a PhD in computer science from the University of Dortmund, Germany. Contact him at [thorsten@joachims.org](mailto:thorsten@joachims.org).*

*Filip Radlinski is a PhD student in the Department of Computer Science at Cornell University. His research interests include machine learning and information retrieval with a particular focus on implicitly collected user data. He is a Fulbright scholar from Australia and the recipient of a Microsoft Research Fellowship. Contact him at [filip@cs.cornell.edu](mailto:filip@cs.cornell.edu).*



**Get access**

**to individual IEEE Computer Society documents online.**

More than 100,000 articles and conference papers available!

*\$9US per article for members*

*\$19US for nonmembers*

[www.computer.org/publications/dlib](http://www.computer.org/publications/dlib)

IEEE computer society