A Mobile Web Focused Search Engine Using Implicit Feedback

Malvika Pimple
Department of Computer Science
University of North Dakota
Grand Forks, ND 58202
malvika.pimple@email.und.edu

Wen-Chen Hu
Department of Computer Science
University of North Dakota
Grand Forks, ND 58202
wenchen@cs.und.edu

Naima Kaabouch
Department of Electrical Engineering
University of North Dakota
Grand Forks, ND 58202
naimakaabouch@mail.und.edu

Hung-Jen Yang
Department of Industrial Technology Education
National Kaohsiung Normal University
Kaohsiung City, Taiwan
hjyang@nknucc.nknu.edu.tw

Abstract

The diverse backgrounds of Web users lead to the notion of personalized search engines. Traditional search engines normally provide the same ranked search results to everyone. The concept of "one ranking fits all" of search engines is not effective, especially when the search is from a personal mobile device. This personalization can be achieved by a machine-learning algorithm, which learns from implicit feedback provided by the users in the form of clicks. This research provides personalized rankings based on this collected implicit feedback data by using support vector machines (SVM). The support vector machine builds a unique model according to the user-feedback data. Search results are ranked according to the unique model to meet each user's specific needs. The proposed system greatly improves the overall ranking quality of search results.

1 Introduction

Data available on the Internet is growing exponentially (Amin, Bhattacharjee, & Jamil, 2010). Since there are no specific standards for the organization and structure of websites and Web pages, finding relevant and up-to-date information without the help of an efficient searching technique can be very difficult. Web search engines thus provide a suitable medium to find specific content from a vast repository of information. As more Websites are practicing search engine optimization (Davis, 2006) and using search engine marketing (Moran & Hunt, 2005), people have to rely on search engines to judge the "relevance" of every page. In addition to the problem of depending on a search engine to judge relevance, Web search engines can also introduce a significant amount of bias to people's perception of the Web (Joachims & Radlinski, 2007). A study shows that, most major search engines use PageRank and its variations (Cho & Sourashis, 2004). Due to this, many high quality pages are being ignored by Web users, simply because no search engine has discovered them yet.

In recent years, personalization has become a popular trend on the Internet. Personalization is a way of providing information organized in a way that is both relevant and personal to each individual user. Personalization is mostly done by providing effective filters and recommendations to the user. With the growing popularity of this feature, personalization has become a key facet of the evolving Web. The concept of personalization has become a requisite for the users of mobile Internet. It has been predicted that in the next decade, mobile Internet will overtake the fixed/desktop Internet (Meeker et al., 2010). This research presents one way of personalizing the search results using implicit feedback data available from users. Two major pieces of software are implemented by the presented research: (i) a Web-focused search engine and (ii) a machine-learning software, which uses a support-vector machine to learn, classify, and rank search results. The focused search engine includes three components:

- The crawler, which is used to collect mobile Web pages by traversing hyperlinks
- The indexing software, which indexes the Web pages collected by crawlers for fast searching and retrievals
- The searching and ranking software, which is used to retrieve and rank the search results

The machine-learning software includes the following components:

- The logger, which keeps track of the queries from users, the ranking information presented to the user for that query, and the clicked documents,
- The support vector machine (SVM), which is used to classify documents, and
- The feature mapper, which maps documents and queries to an n-dimensional feature space.

The proposed method is described as follows. When a user performs a search, the logger records all the information about the search query q, retrieved documents d_i, and their computed ranks r_i. The feature mapper is then used to extract predefined features which are unique to (q, d_i, r_i). The extracted features are stored in the log. If the user clicks on some document d in d_i, the logger records the click for (q, d, r). Such

recorded clicks are treated as preference feedback. Features from logged data along with preference feedback data are used to train SVM to produce a unique model. This model is used to perform classification against test examples (all future user searches). When a new search is submitted, the extracted features from the retrieved set of documents are passed to SVM as test examples for ranking. The empirical results show that the proposed method of using implicit user feedback improves search experience and greatly enhances the overall ranking quality of mobile search results.

The rest of this paper is organized as follows. Section 2 gives a brief literature review of existing research in the area of WWW search engines and implicit feedback. Section 3 describes the mobile focused search result implemented for the purposes of this research. Section 4 describes support vector machines and its implementation for this research. Section 5 presents experimental results and gives analyses and comparisons of presented research with existing generic search engines. Section 6 concludes this paper and gives future directions.

2 Related Research

Web search engines—Google, Bing and others—are by far the most popular and heavily used information retrieval (IR) services, providing access to up-to-date information. A generic WWW search engine consists of the following modules:

- *Information gathering*: Information gathering is more specifically known as Web crawling. The crawlers are programs that scan Web pages in a methodical manner to gather information from the content and structure of Web pages. The crawler maintains a list of visited pages. How the visited page is processed can be modified and enhanced depending upon different application areas and type of Web page
- *Data extraction and indexing*: Data extraction and indexing is the process of algorithmically examining information items to build a data structure that can be quickly searched. B-trees and inverted files are two common data structures for information retrieval. This module creates indexes for URLs by using keywords, links, titles, etc. This module in turn consists of the following sub modules:
    - *Text acquisition:* Identifies and stores documents for indexing. Text, metadata and other related content for the document is stored
    - *Text transformation:* Transforms documents into index terms or features. This module consists of parser which tokenizes the words, stemming module which processes grouping of words derived from common stem (for example, “computer”, “computers”. “computing”, “compute”, etc.), a link analyzer which identifies popularity information of the links, an information extractor which identifies important terms for some applications and a classifier which identifies class-related metadata for documents
    - *Index creation:* Takes index terms and creates data structures to support fast searching. It carries out statistical analysis which is used to rank documents. It also assigns term weigh to the index term. The term weight is usually a function of term-frequency. This module stores the extracted terms and their scores in an inverted index. This module can also distribute the index across several computer or site for optimization purposes

- *Searching and Ranking module:* Searching and ranking module provides the basic interface for the user to query the search engine. This module consists of the following units: a user interaction unit which provides an interface for search queries, a query transformation which improves initial query and provides auto suggestions and spell checking and a result output which displays a ranked list of documents for a query. The ranking module uses a query, log and index to generate a ranked list of document. The ranking algorithm is one of the most important components of any search engine. This module performs scoring, performance optimization and evaluation tasks.

Much research has been done in the area of personalization using implicit user feedback. Since most users would not bother to provide explicit feedback, the trend is to focus on implicit feedback (Joachims, 2005). Implicit feedback can be effectively used to reveal a user's search intentions (Fox, Karnawat, Mydland, Dumais, & White, 2005). Research also shows that there is a lack of significant difference between the 'implicit' and 'explicit' systems of feedback (White, Jose, & Ruthven, 2001). There are many ways to gather implicit feedback and use it effectively. A few approaches are query history, click data, display time/attention time, exit type/end action, etc. Query history is the most used implicit user feedback by search engines including Google (http://www.google.com/psearch). Click data is another implicit feedback which is readily available and can be collected at a relatively low cost. Research has been done to show that click through data can be accurately interpreted in the form of relative-feedback rather than absolute feedback (Joachims, Granka, & Pan, 2005). Display time/user attention time is the amount of time a user spends to read a particular document. At present, no browser or application exists which can provide accurate attention time of a user on a Web page. Exit type/end action shows how a user exited a result or ended a search session. Research shows that when users spend 58 seconds on a page and do not go back to the search result, they are satisfied 88% of the time (Fox, 2005). When users spend very little time on a page (<58 seconds), and whey they do go back to the search result, they are likely to be dissatisfied 73.4% of the time (Fox, 2005).

Research has also been done in the area of how implicit feedback can be effectively used. A method presented by Lv, Sun, Zhang, Nie, Chen, & Zhang (2006) implements result re-ranking and query-expansion simultaneously and collaboratively. It uses a client-side personalized Web search agent PAIR (Personalized Assistant for Information Retrieval) which implements a HITS-like (Hyperlink-Induced Topic Search) iterative approach for personalized search. Another study presents a user oriented Webpage ranking algorithm which is based on the user attention time (Xu, Zhu, Jiang, & Lau, 2008). The ranking algorithm employs user's attention time to produce a user-oriented Web page ranking based on predictions. The prediction algorithm employs document content similarity analysis, which is applied to both text and images. The experimental results prove that the algorithm satisfactorily produces user-oriented Webpage ranking. Another method uses click-through data and support vector machine (SVM) approach to optimize search engines (Joachims, 2002). The method makes use of the strong dependencies between the query, rank and clicked-document. The algorithm uses SVM for learning ranking

functions and producing a ranking (r^f) which is close to the optimal ranking (r^*) in which highly relevant documents are ranked higher than other documents.

The proposed system implements a focused search engine which performs crawling and indexing of limited number of pages under a given seed URL. This focused search engine records user search history along with implicit feedback data in form of user clicks on search results. These clicks are treated as preference statements and given as input to the support vector machine. The support vector machine builds a unique model based on user search history. This unique model is used for ranking search results for all future searches. The next chapter describes the implemented focused search engine.

3 The Mobile Focused Search Engine

This section gives a detailed description of the proposed system structure. Figure 1 shows a block diagram of system structure and functionality. The system includes the following sub-components: (i) crawling unit, (ii) indexing unit, (iii) search and ranking unit and (iv) support vector machine unit. The support vector machine will be described in more detail in Section 4.

The crawling unit (also called spider or robot), is a program that automatically scan various Web sites and collects the Web documents from them by follow the links on a site to find other pages. The crawling unit of the proposed system crawls the web and finds n unique URLs which strictly fall under seed URL domain. These n unique URLs are then passed on to indexing unit. The indexing unit takes an unvisited URL from the list, performs parsing and extracts useful information in the form of keywords and their ($tf.idf$) scores and stores the information in datastore. This process can only run for t minutes. The sequence of processes carried out for searching and ranking is illustrated using numbers in Figure 1. When a user queries (q) the search engine, the searching and ranking unit retrieves the top 10 documents $\{d_1, d_2, \ldots, d_{10}\}$ from the datastore ordered by descending ($tf.idf$) scores. Before presenting the retrieved information to the user, the searching and ranking unit logs the information (q, di, r, f_1, $f_2, \ldots, f_n$) where r is the rank of document d_i and f_1-f_n are features derived from 3-tuple (q, d_i, r). Feature calculation is described in more detail later in this paper. The retrieved list of documents is then presented to the user. The ranking of the presented documents may be different from the descending order of ($tf.idf$) ranking. When user feedback is available in the form of clicks on search results, the searching and ranking unit implicitly collects and logs this information. This logged information is used to train the SVM module to extract user preferences and generate a model specific to user feedback data. The next section gives some background of support vector machines and how they are used in this research.

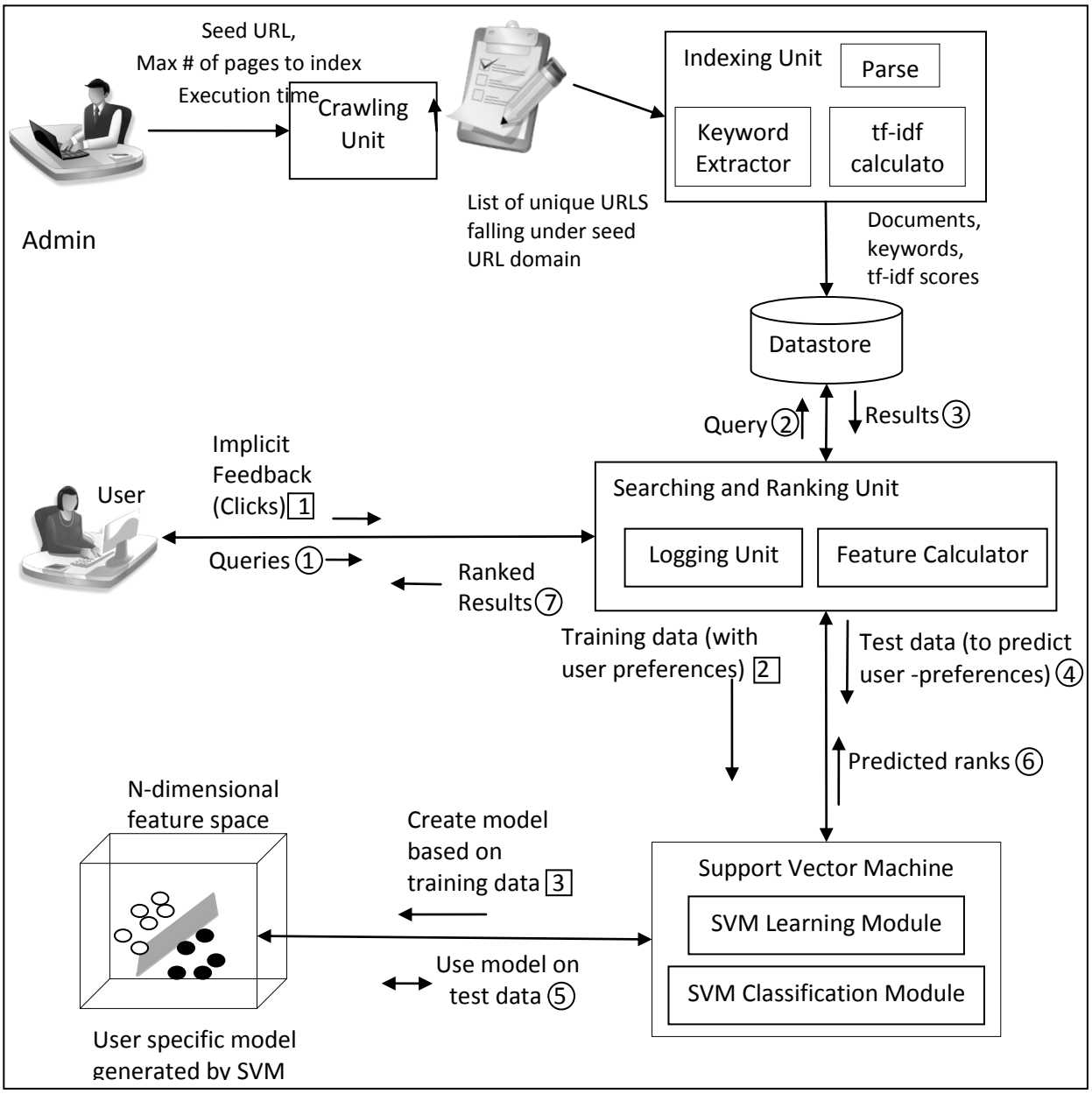

Figure 1: A Block Diagram of the Proposed System.

4 Support Vector Machines

Support vector machines (SVMs) are currently a hot topic in the machine learning community (Meyer, 2010). Support vector machines (also known as support vector networks) were developed by Cortes and Vapnik (1995). An example of the two-group classification is shown in Figure 2.

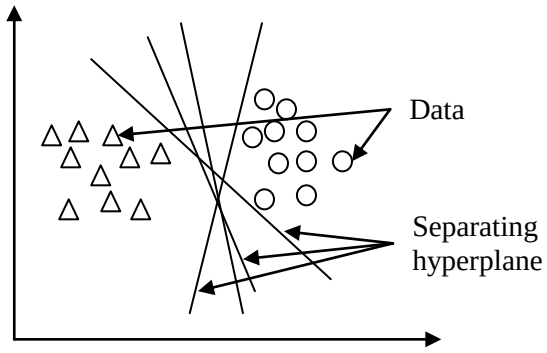

Figure 2: Possible Separating Hyperplanes for Binary Classification.

Consider *l* training examples $\{x_i, y_i\}$, $i=1,\ldots,l$, where each example has d inputs $(x_i \in R^d)$, and a class label with one of the two values $y_i \in \{-1,1\}$. Now all hyperplanes in R^d are parameterized by vector (w), and a constant (b), expressed in the equation:

$$w \cdot x + b = 0 \tag{1}$$

w is infact the vector orthogonal to the hyperplane. Given that this hyperplane (w,b) seperates data, gives the function:

$$f(x) = \text{sign}\,(w \cdot x + b) \tag{2}$$

This function correctly classifies the training data (and other "testing" data). However, a given hyperplane represented by (w,b) is equally expressed by all pairs $\{\lambda w, \lambda b\}$ for $\lambda \in R^+$. The canonical hyperplane is defined to be that which seperates the data from the hyperplane by a distance of at least 1. That is, it should satisfy the following:

$$x_i \cdot w + b \geq +1 \text{ when } y_i = +1 \tag{3}$$

$$x_i \cdot w + b \leq -1 \text{ when } y_i = -1 \tag{4}$$

Or more compactly:

$$y_i(x_i \cdot w + b) \geq 1 \quad \forall i \tag{5}$$

All such hyperplanes have a "functional distance" ≥ 1 (not to be confused with "geometric" or "Euclidean distance". This is also known as margin). For a given hyperplane (w,b), all pairs $\{\lambda w, \lambda b\}$ define the exact same hyperplane, but each has a different functional distance to a given data point. To obtain the geometric distance from the hyperplane to a data point, we must normalize by the magnitude of w. This distance is simply:

$$d((w,b),x_i) = \frac{y_i(x_i \cdot w + b)}{\|w\|} \geq \frac{1}{\|w\|} \tag{6}$$

Intuitively, we want the hyperplane that maximizes the geometric distance to the closest data points. These closest data points are also known as support vectors. From the equation, we see that his is accomplished by minimizing $\|w\|$ (subject to the distance constraints). This minimization problem is further transformed into what is known as a Quadratic Programming Problem (QP). Fortunately many techniques have been developed to solve them.

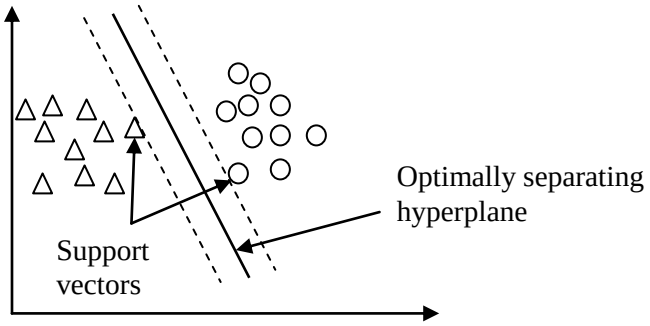

Figure 3: Support Vector Machine for Binary Classification.

SVM^{rank} (Joachims, 2006) is a support vector machine algorithm for solving such classification problems. SVM^{rank} consists of a learning module (`svm_rank_learn`) and a module for making predictions (`svm_rank_classify`). The ranking module trains a Ranking SVM on the training set and outputs the learned model. To make predictions on test examples, `svm_rank_classify` reads this model and the predicted ranking score is written to a file. SVM^{rank} learns an unbiased classification rule. It also supports kernels. To use the support vector machine algorithm, one must map the data elements to be classified onto an n-dimensional feature-value space. How data from the search engine is mapped onto a multi-dimensional feature-value space is discussed in the following subsection.

4.1 Feature mapping

For a query q and a document d, a feature mapping function is defined represented by $\Phi(q,d)$ which describes the match between the query `q` and document `d`. Some features are for example, the number of words that query and document share, the number of words they share inside certain HTML tags (e.g. TITLE, H1, H2, …), or the page-rank of `d`. For the proposed system, the following feature mappings are implemented:

1. *Query-URL Cosine:* Cosine between URL-words and query (range `[0,1]`)
2. *Query-Abstract Cosine:* Cosine between title-words and query (range `[0,1]`)
3. *Query-Document Frequency:* Term-frequency of query in document
4. *Domain name in query:* Query contains domain name from URL (binary `{0,1}`)
5. *URL-length/30:* length of URL in characters divided by 30
6. *Initial rank of document in search results*: Initial rank of the document before classification

Cosine similarity is a measure of similarity between two vectors of n dimensions by finding the cosine of the angle between them. Two vectors in two-dimensional space are shown in Figure 4.

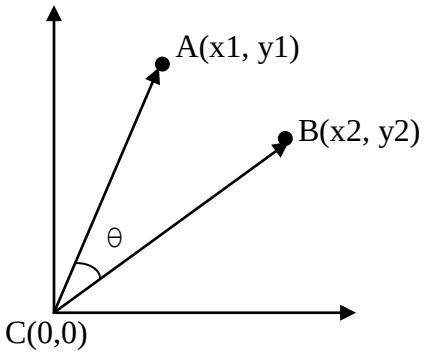

Figure 4: The Cosine Angle between Vectors A and B.

The cosine of the angle between the two vectors is given by:

$$\text{cosine } \theta = \frac{A.B}{|A||B|} \tag{7}$$

This ratio is also used as a similarity measure between any two vectors representing documents, queries or combinations of these.

$$\text{Sim}(A,B) = \text{cosine } \theta = \frac{A.B}{|A||B|} = \frac{x1*x2+y1*y2}{(x1^2+y1^2)^{1/2}(x2^2+y2^2)^{1/2}} \tag{8}$$

As the angle between the vectors shortens, the cosine angle approaches 1, meaning that the two vectors are getting closer. This measure is convenient way of measuring text similarity (Tata & Patel, 2007). The documents or queries or any text can be represented as points in an n-dimensional term space. The term space is defined by a list (index) of terms. These terms are extracted from the collection of documents to be queried. The coordinates of the points representing documents and queries are defined according to the weighting scheme used. In the case of our proposed system, the weights are defined as term-frequencies. Thus in context of information retrieval, the similarity equation is defined as:

$$\text{Sim}(Q, Di) = \frac{\sum_j w_{Q,j} w_{Di,j}}{\sqrt{\sum_j w_{Q,j}^2}\sqrt{\sum_j w_{Di,j}^2}} \tag{9}$$

Where, Q is a query, D is a document and w weights $(w = tf)$. The next section describes the experimental setup and analyses carried out to check whether the support vector machine learning algorithm successfully learns from the user-feedback data.

5 Experimental Results and Analyses

For experimentation purposes, a mobile focused search engine was developed as described in Section 3. This implemented mobile search engine provides basic search interface for the user to query the search engine and an admin interface for the administrator to control the number of pages and Website to be indexed. The interfaces for searching, indexing and administration are shown in Figure 5. Figure 6 shows an example of comparison of search results with and without learning component turned ON for the query "calendar."

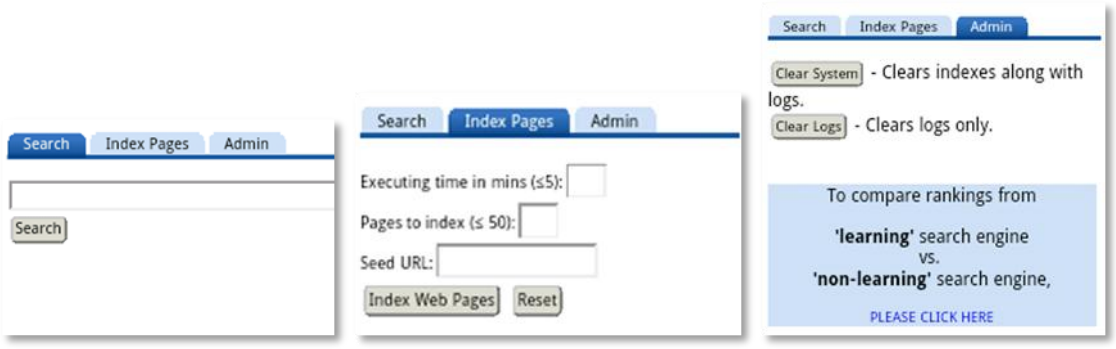

Figure 5: Interfaces for Searching, Indexing and Administration.

For experimental purposes, eight small Websites were chosen to be indexed by the proposed system. Table 1 shows the number of pages indexed by the proposed system versus the number of pages indexed by Google. The numbers for Google were obtained from UND's official Google search appliance reports.

Websites	The Proposed System	Google
http://workwell.und.edu	46	129
http://go.und.edu	75	181
http://und.edu/org/finaid	50	115
http://wellness.und.edu	49	226
http://cs.und.edu	69	106
http://career.und.edu	32	170
http://tvcenter.und.edu	19	56
http://dining.und.edu	40	70
Total	380	1053

Table 1: Examples of Numbers of Pages Indexed.

The main mission of this research is to predict preferred ranking for a user based on user's click-through data. Since user clicks on only a subset of documents from a ranked list of documents, the system logs the clicked document and the number of clicks made on the particular document. From each recorded click, a preference ranking can be extracted. Using such preference rankings we create a target ranking which represents user's preferred ranking. The goal of this research is to produce a ranking r which is closer to user's preferred target ranking r^*. To compare rankings the measure Kendall's τ was used which is described in the next subsection.

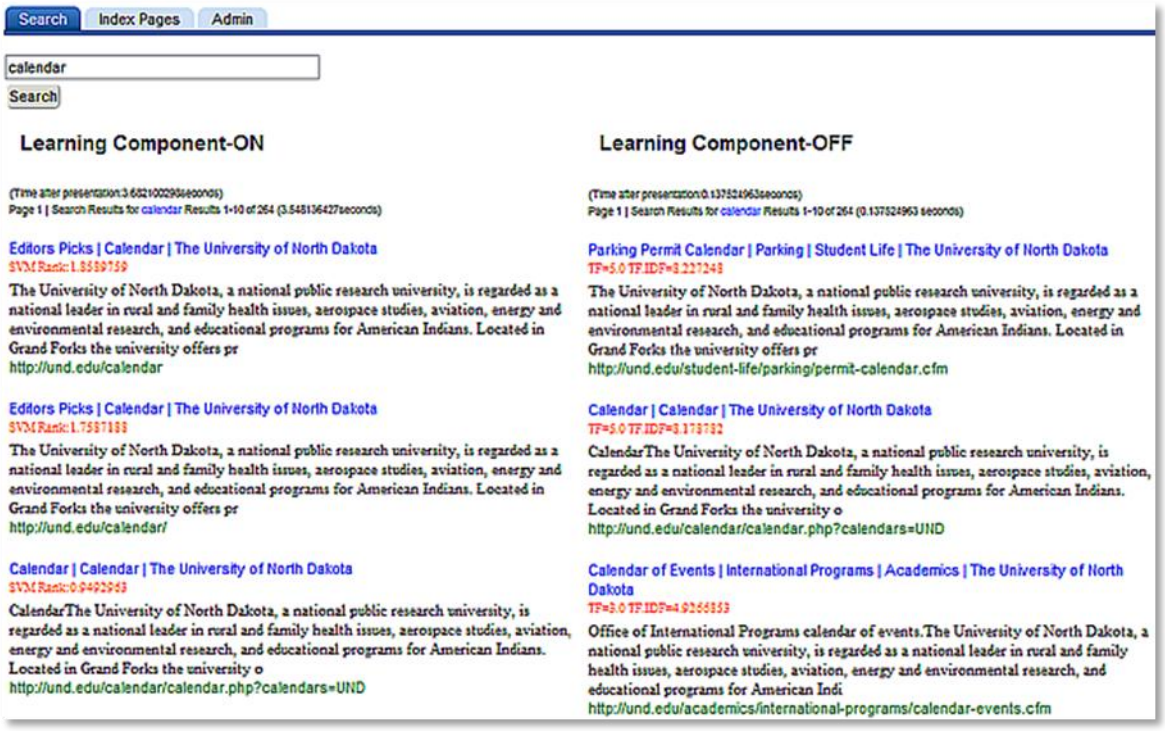

Figure 6: Comparison of Search Results with or without Learning Component ON for the Query "calendar."

5.1 Rank Correlation Analysis Using Kendall's τ

Kendall's τ is the measure of rank correlation. It measures similarity of orderings of data ranked by two separate quantities. For two strict orderings $r_a \subset D \times D$ and $r_b \subset D \times D$, Kendall's τ can be defined based on the number P of concordant pairs and the number Q of discordant pairs (inversions). A pair (d_i, d_j) where, $d_i \neq d_j$ is concordant, if both r_a and r_b agree in how they order d_i and d_j. It is discordant if they disagree.

For a finite domain D of m documents, the sum of P and Q is $\binom{m}{2}$ for strict orderings.

$$P + Q = \binom{m}{2} = \frac{m!}{2!\,(m-2)!} \tag{10}$$

In this case, Kendall's τ can be defined as:

$$\tau(r_a, r_b) = \frac{P-Q}{P+Q} = 1 - \frac{2Q}{\binom{m}{2}} \tag{111}$$

Table 2 shows an example of log file records for a query 'exam'. The generated results are shown in the table along with which links were clicked, total number of clicks on that URL, the static ranking of that URL, and the ranking predicted by our learning component.

D#	URLs	Clicked	# of Clicks	tf.idf Rank	Predicted Rank	Target Rank
d_1	http://cs.und.edu/Graduate/GeneralInformation.aspx	No	0	1	9	3
d_2	http://cs.und.edu/Graduate/GradCompExam.aspx	Yes	2	2	1	1
d_3	http://cs.und.edu/sitemap.aspx	Yes	1	3	2	2
d_4	http://cs.und.edu/Graduate/FinancialInformation.aspx	No	0	4	8	-
d_5	http://cs.und.edu/Graduate/Default.aspx	No	0	5	4	-
d_6	http://cs.und.edu/Graduate/PriorGraduateWork.aspx	No	0	6	7	-
d_7	http://cs.und.edu/Graduate/ResearchOpp.aspx	No	0	7	6	-
d_8	http://cs.und.edu/Graduate/PhD.aspx	No	0	8	3	-
d_9	http://cs.und.edu/Graduate/Masters.aspx	No	0	9	5	-

Table 2: Target Ranking versus Predicted Ranking for Query "exam."

From the click data available, we can create a 'target ranking' r^*, which is the user's preferred ranking based on the clicks. For example, from the log data shown in Table 2, user clicked on the document d_2 twice and d_3 once with no clicks on d_1. From this statement, the following user preference can be extracted: 'd_2 should be ranked higher than d_1'. This can be represented as

$$d_2 <_{r^*} d_1 \tag{12}$$

Similarly, the following preference statements can be extracted from the table.

$$d_3 <_{r^*} d_1 \tag{13}$$

$$d_2 <_{r^*} d_3 \tag{14}$$

The resulting target ranking is represented in the last column of Table 2. All the other documents, for which a preference statement cannot be extracted, are ignored. Using that as a target ranking, we now calculate Kendall's τ, the rank correlation for $tf \cdot idf$ ranking

r_s versus r^*, and predicted ranking r_p versus r^* to see if our learning component produces results closer to the r^* as compared with the static ranking based on $tf \cdot idf$.

5.1.1 Kendall's τ for r_s versus r^*

The target ranking (r^*) extracted for the given data is:

$$d_2 <_{r^*} d_3 <_{r^*} d_3 \tag{15}$$

While the static $tf \cdot idf$ ranking (r_s) for documents d_1, d_2 and d_3 can be represented as:

$$d_1 <_{r_s} d_2 <_{r_s} d_3 \tag{16}$$

The number of discordant pairs Q are 2: {(d1, d2), (d1, d3)} and number of concordant pairs P is 1 {(d2, d3)}. So Kendall's τ can be calculated as:

$$\tau = \frac{P-Q}{P+Q} = \frac{1-2}{3} = -0.33 \tag{17}$$

5.1.2 Kendall's τ for r_p versus r^*

From Table 2, the predicted ranking $(r_p)$for d_1, d_2 and d_3 is:

$$d_2 <_{r_p} d_3 <_{r_p} d_1 \tag{18}$$

Number of discordant pairs Q between (15) and (16) is 0 and number of concordant pairs is 3. So Kendall's τ is 1.

$$\tau = \frac{P-Q}{P+Q} = \frac{3-0}{3} = 1 \tag{19}$$

From the analyses and comparisons presented in this chapter, we can see that the predicted ranks generated using our system are better than the traditional $tf \cdot idf$ ranks. It can be seen that the implemented system successfully learns user preferences based on clicks without any explicit feedback or manual parameter tuning.

5.2 Analysis of Number of Search Results and Search Speed

Figures 7 and 8 show the histogram of number of search results comparision and search speed comparison of the proposed search engine with Google, Bing and Yahoo generic search engines. The results show that the number of search results returned by the proposed search engine is comparable to those of Google, Bing and Yahoo.

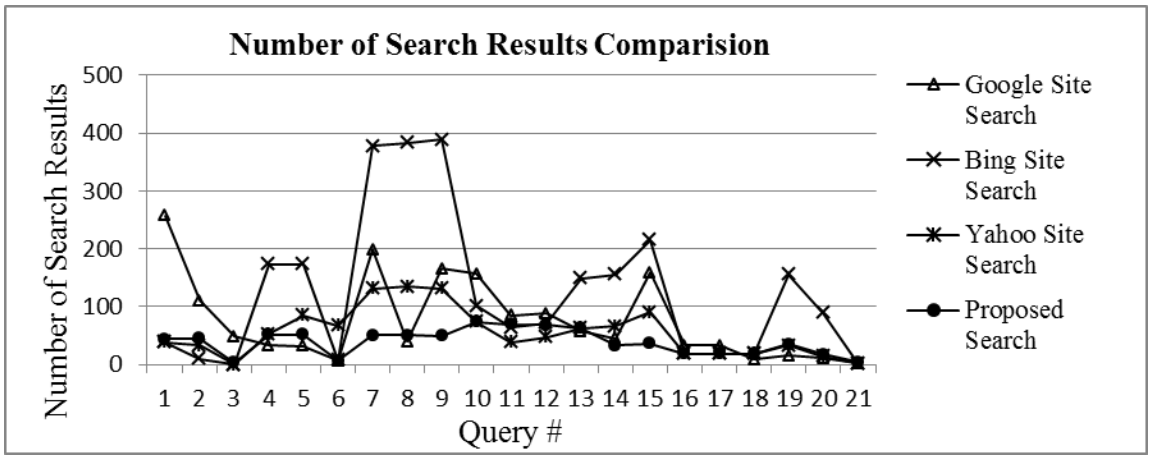

Figure 7: A Histogram of Search-Result Numbers of Search Engines.

For search speed analysis of our proposed search engine, the search time is defined as—the time between query submission and the time when the browser finishes displaying the results. Not all search engines have search time implemented in their system. Among the other search engines, Google is the only one which displays its search speed. As it can be seen from the results shown in Figure 8, our proposed search engine with learning component turned off performs a lot faster than Google™ for all cases. However, with learning component turned on, the performance deteriorates a little.

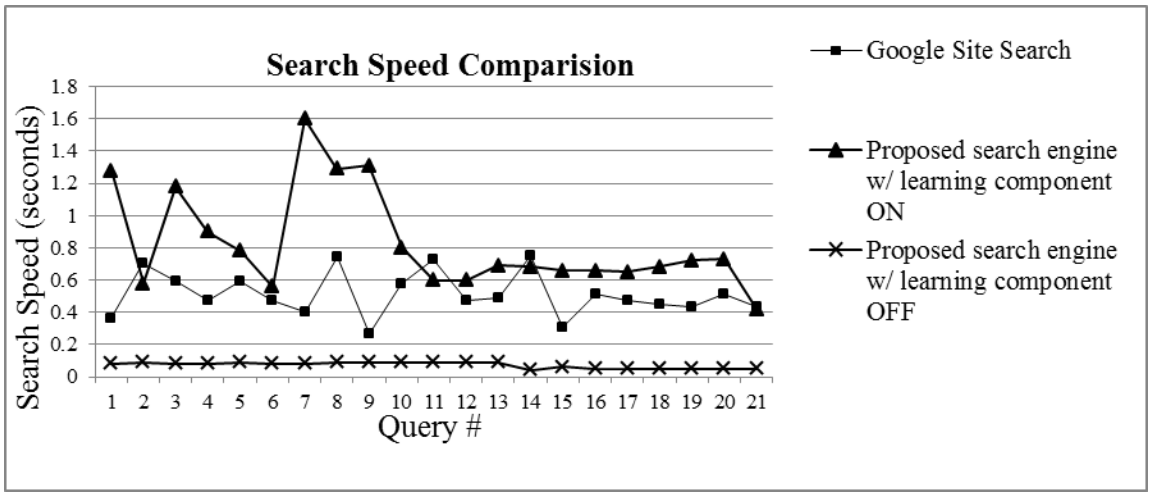

Figure 8: Search Speed Comparison Histogram.

6 Conclusion

With growing popularity of smart phones and personalized mobile applications, personalization of search results has also become a necessity. This research proposes a possible way of implementing a search engine that learns from implicit user feedback to produce search results that are personalized. This personalization is done by a machine

learning algorithm that learns user preferences and builds a model. The search engine then classifies the retrieved results and re-ranks them using the model as an input.

To demonstrate how clicks are used to generate personalized ranking, this research implements a simple focused search engine. Though it cannot be compared with major commercial search engines without any prior adjustments, an effort has been made to achieve the basic search engine functionality to a maximum extent. The comparisons and analysis show that learned function improves retrieval as compared to a static function. The presented algorithm is automated and does not require any adjustments. Along with several advantages, this system also has some disadvantages. The system relies on implicit feedback obtained from clicks which can be inherently noisy. A few unintended clicks are also considered as relevance feedback which can have a significant impact on search results. Also, since this system uses extra steps to re-rank results, the turn-around time is slightly greater than most of the generic search engines.

Although the proposed methods are implemented for Web search engines in this research, the idea can be extended for other software applications. Implementation of more features may significantly enhance overall user experience. The future work may include the following:

- Other types of implicit feedback like user time on page and entry/exit type can be used to enhance the personalization of search results.
- Relevance feedback can be used to generate query reformulations and autosuggestions.

References

Amin, M. S., Bhattacharjee, A., & Jamil, H. (2010). Wikipedia driven autonomous label assignment in wrapper induced tables with missing column names. In *Preceedings of the 2010 Symposium on Applied Computing (SAC '10)*, pages 1656-1660, Sierre, Switzerland.

Cho, J. & Sourashis, R. (2004). Impact of search engines on page popularity. In *Proceedings of the 13th International Conference on World Wide Web*, New York, USA.

Cortes, C. & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, vol. 20, no. 3, pp. 273-297.

Davis, H. (2006). *Search Engine Optimization: Building Traffic and Making Money with SEO.* O'Reily Media, Inc.

Fox, S., Karnawat, K., Mydland, M., Dumais, S., & White, T. (2005). Evaluating implicit measures to improve Web search. *ACM Transactions on Information Systems,* vol. 23, no. 2, pp. 147-168.

Joachims, T. & Radlinski, F. (2007). Search engines that learn from implicit feedback. *Computer*, vol. 40, issue 8, pp. 34-40.

Joachims, T. (2002). Optimising search engines using clickthrough data. In *Proceedings of 8th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD 02)*, Edmonton, Alberta, Canada.

Joachims, T. (2006). Training linear SVMs in linear Time. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD 06)*, Philadelphia, Pennsylvania, USA.

Joachims, T., Granka, L., & Pan, B. (2005). Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retreival (SIGIR '05)*, Salvador, Brazil.

Lv, Y., Sun, L., Zhang, J., Nie, J.-Y., Chen, W., & Zhang, W. (2006). An iterative implicit feedback approach to personalized search. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL*, pages 585-592, Sydney, Australia.

Meeker, M. et al. (2010). *Internet Trends*, Morgan Stanley. Retrieved January 21, 2011, from http://www.morganstanley.com/institutional/techresearch/pdfs/Internet_Trends_041210.pdf

Meyer, D. (2011). *Support Vector Machines: The Interface to libsvm in Package e1071*. Retrieved March 12, 2011, from http://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf

Moran, M. & Hunt, H. (2005). *Search Engine Marketing, Inc.: Driving Search Traffic to Your Company's Web Sites.* IBM Press.

Tata, S. & Patel, J. M. (2007). Estimating the selectivity of tf-idf based cosine similarity predicates. *SIGMOD Record*, vol. 36, no. 2, pp. 7-12.

White, R. W., Jose, J. M., & Ruthven, I. (2001). Comparing explicit and implicit feedback techniques for Web retrieval: TREC-10 interactive track report. In *Proceedings of the 10th Text REtrieval Conference (TREC 2001)*, Gaithersburg, Maryland, USA.

Xu, S., Zhu, Y., Jiang, H., & Lau, F. C. M. (2008). A user-oriented Webpage ranking algorithm based on user attention time. In *Proceedings of the 23th AAAI Conference on Artificial Intelligence*, Chicago, USA.