Mobile Web Search Personalization Using Ontological User Profile

Kapil Goenka
Department of Computer Science
University of Georgia
Athens, GA 30602-7407
kapil.goenka@gmail.com

I. Budak Arpinar
Department of Computer Science
University of Georgia
Athens, GA 30602-7407
budak@cs.uga.edu

Mustafa V. Nural
Department of Computer Science
University of Georgia
Athens, GA 30602-7407
mvnural@cs.uga.edu

ABSTRACT

Most present day search engines have a deterministic behavior in the sense that they return the same search results for all users who submit the same query at a certain time. They do not take the user's interests and preferences into account in the retrieval process. Integrating user context in the retrieval process can help deliver more targeted search results, thereby providing a personalized search experience to the user. Personalizing web search involves the process of identifying user interests during interaction with the user, and then using that information to deliver results that are more relevant to the user. In this paper, we present our approach to personalizing web search on a mobile device (iPhone). Our approach involves building an ontological model of user interests on the user's mobile device based on his interaction with web search results. Personalization of search results is achieved by re-ranking search results returned by a standard search engine (Yahoo) based on proximity to the user's interest model. The ability to recognize user interests in a completely non-invasive way and the accuracy of personalized results are some of the major advantages of our approach.

Keywords

Search Personalization, User Profiles, Sematic Web, iPhone.

1. INTRODUCTION

Today, internet search engines have become an indispensable part of our lives. People today are able to find all sorts of information instantly from almost anywhere. The exceedingly difficult nature of the problem of understanding user intent and matching it with the world's accumulated knowledge stored on the World Wide Web has attracted large scale research and development efforts from the academia as well as the industry. In the recent years, we have also seen an explosive growth in mobile devices. The modern cell phones are significantly better than the one's from a few years ago. Mobile Internet has quickly become part of the consumer media experience for millions of people. More people are searching the web while they are on the move.

Although the capabilities of Internet search engines are incrementally improving, there are several challenges facing the search engines. One challenge is the problem of irrelevant search results. Irrelevant search results usually arise due to short, ambiguous queries or semantic level mismatches. Examples include "apple", "Pascal", "match", "conductor" etc. all of which can have different meanings depending on context. Another cause for irrelevant results is the one-size-fits-all approach taken by most existing search engines, where an identical query from different users in different contexts will generate the same set of results for all users. These search engines return a list of search results based on a user's query but ignore the user's specific interests, search context and individual differences in information needs. As a result, a user may have to go through many irrelevant results before finding the desired information. Mobile web search introduces new challenges not present in traditional web search. The input modes are inherently limited due to the small size of the device itself and the network connectivity is often not comparable to the Internet speeds on computers. Mobile users are likely to be on the go when searching for information and the attention span of the users is significantly lower than in traditional web search on computers. Furthermore, the user is unlikely to sift through a lot of search results to get to the desired page due to his short attention span. It is therefore very important to get the desired search results in the top positions to avoid waste of time and effort for the user.

Personalization techniques that incorporate user interests and preferences into the search may address some of these issues. Personalization broadly involves the process of learning a profile of user interests. Personalization of web search usually involves filtering or re-ranking the results returned from a standard search engine, or directly incorporating user interests into the retrieval process itself to present personalized results. Given a query, a personalized search can provide different results for different users or even different results for the same user in different contexts. Web search personalization has two main dimensions:

1. How can precise information about user's interests be collected and represented?
2. How can this information be used to deliver personalized search results?

In this work, we present our approach to personalizing web search in a mobile environment. As a case study, we chose Apple's iPhone as the mobile platform to implement our work. Our main goal is to identify user's interests based on the web pages he visits, and deliver personalized web search results by utilizing the identified user interests. We learn and maintain implicitly an ontological profile of user's interests through passive observation of the user's click stream. The user's interest profile is stored locally on his mobile device and updated with every web page visit. Personalization is achieved by re-ranking standard web search results using the user's interest profile.

2. BACKGROUND

2.1 Web Directories

Web Directories, also referred to as knowledge bases, are a popular means of organizing information resources on the web. A web directory is a repository of web pages that are organized in a hierarchical structure, usually like a tree or a directed acyclic graph (DAG). Each web page cataloged in a web directory is annotated with a short description by one of the editors of the directory. Many web directories have become available in recent years. The Librarian's Internet Index (LII) [2], The Internet Public Library (IPL) [3], Yahoo Web Directory [4] and the Open Directory Project (ODP) [5] are examples of general purpose web directories. These web directories catalog huge numbers of URLs organized in an elaborate hierarchy. Since the actual process of creating such ontologies can be a very tedious, most hierarchical classification systems utilize existing web directories as their predefined class hierarchies.

2.1.1 Open Directory Project (ODP)

In this work, we use the Open Directory Project (ODP) as our knowledge base. ODP is one of the largest collaborative efforts to manually annotate web pages and is widely regarded as "the largest human-edited directory of the web". Currently ODP catalogues over 4.6 million URLs that have been categorized into nearly 600,000 categories by over 80,000 human editors. ODP's data structure is organized as a DAG. The textual data contained in the leaf nodes can be utilized as training data for the parent concept of the leaf node. Web directories like ODP cover most, if not all, information domains, and can therefore be used for representing user interests. Nodes at the top levels of the hierarchy represent broad user interests and the ones below them narrow down the scope of their ancestors. In this work, we select a subset of concepts from the top four levels of ODP for representing user interests. We focus on the top levels of the hierarchy since we believe that many search results can be usefully disambiguated at this level.

2.2 Text Classification Using Rainbow

We use the open source Rainbow text classification library [7] by Andrew McCallum at CMU as the "kernel" of our text classification module. The Rainbow Text Classifier, is perhaps the most well known and most downloaded text classifier today. It supports a number of text classification methods for classifying text into a set of topics. Rainbow must be trained before using it for classification. This involves creating a model of a set of training documents. The training set is read in as directories (one per category) containing text files that serve as examples for those categories. Once Rainbow is trained, it can be set up as a server that received classification requests over a port. After a model is learnt from the training set, classification can be performed using one of the many classification methods supported by Rainbow (Näıve Bayes, Term Frequency - Inverse Document Frequency (TFIDF), probabilistic indexing, k-nearest neighbor and support vector machines (SVMs)).

In this work, we train the rainbow classifier on a subset of the first four levels of the ODP categories and set the classifier to be run as a continuous background server process. The classifier listens for document classification requests over a port.

2.3 Yahoo BOSS API

Yahoo BOSS (Build Your Own Search Service) is an open platform that offers programmatic access to the Yahoo Search indices via an API. As of this writing, the Yahoo BOSS API is offered free of charge to developers. There is no limit to the number of queries that can be made. However, a maximum of 50 search results can be fetched per query. The search API allows developers to specify the start position of search results. So fetching the top 500 search results for a query would involve sending 10 API requests, starting with a start position of 0 and incrementing the start position by 50 with each request. We use the BOSS Mashup Framework [8] -- a Python library provided by Yahoo to access the Yahoo search results. We note that other search engine APIs can be used for retrieving standard search results, in pace of Yahoo API. We decided to use the Yahoo API mainly because it provides us access to "key terms" for each search result. Key terms are keywords Yahoo's search index has assigned to a page. It is a finite list of words that explain what a document is about and allow for better categorization. The key terms are obtained by Yahoo based on each term's frequency, and positional attributes in the document. Key terms are particularly useful in our work, as they save us valuable post-processing time, which would otherwise be required for processing result pages and obtaining the key words representing each page. For the purpose of classifying web search results, we consider the combination of key terms, title and snippet of each search result as sufficient information for representing what the web page is about.

3. METHODOLOGY

In the previous sections, we gave some background about the Open Directory Project [5], text classification, and Rainbow text classification library [7]. In this section, we present our methodology to put these components together for personalizing web search on a mobile device.

3.1 Programmatically Accessing ODP

We start with a MySQL database dump of ODP, published in [1]. The MySQL dump provides us a convenient SQL interface to the entire ODP hierarchy. The database contains several tables that together capture all the information in the ODP hierarchy. We query the database using a Java program that connects to the database through a MySQL JDBC connector. Using the Java program, we create a directory structure on the local file system that replicates the ODP hierarchy. The procedure for doing this is shown in fig (1). The output of the procedure is a directory structure where each directory represents a concept. Each directory contains subdirectories representing its sub-concepts, and a Super Document containing the title and description of every web page categorized under that concept.

```
PROCEDURE: CREATE_ODP_STRUCTURE
For every category C in the ODP hierarchy:
  path <--- path of C in the ODP hierarchy
  numSubTopics <--- Number of subcategories of C
  numLinks <--- Number of web pages under C
  if numSubTopics == 0 AND numLinks == 0
    Ignore category C
  else
    Create directory for category C at location 'path'
    if numLinks > 0
    Create Super Document SDc
    For every web page W categorized under C:
        Add the Title & Description of W to SDc
    Save SDc in C's directory
```

Fig 1: Procedure to replicate ODP on a local drive

3.2 Removing Structural Noise from ODP

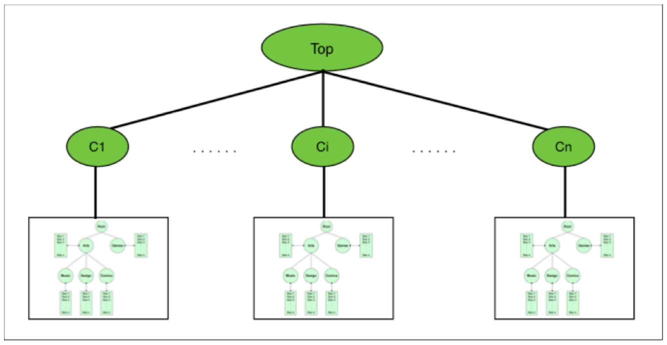
However elaborate knowledge repositories are, they contain concepts that are detrimental to feature generation [6]. These include concepts too deep in the hierarchy, or having too few textual objects to build a representative attribute vector. In [6], they have identified potential sources of noise in ODP. In our work, we use their findings to prune the Top/World, Top/Adult, Top/Kids_And_Teens and Top/Regional branches of ODP:

3.3 Training a Text Classifier on ODP

As discussed in section 2.2, we use the Rainbow text classification library to train a flat multi-label text classifier on a subset of categories from the top four levels of ODP. We first flatten the top four levels of ODP, i.e., bring all categories from those levels under a common parent. We therefore carefully remove all the categories that do not make sense individually. The text classifier is central to our personalization system. Sub-optimal classification results will lead to an inaccurate user model, which may eventually cause irrelevant search results to be returned to the user. We therefore designed a number of experiments around the text classifier. These experiments and their results are shown in section 5. In this section, we present our approaches for training a classifier.

APPROACH I:

1. Select a subset 'S' of concepts from ODP to be used in user modeling.
    a. S = {C1, C2, C3, C4, C5, C6 ...... Cn}
2. Since the concepts in S are from different levels in ODP, flatten them, i.e., move them (along with their sub-trees) to a common level. This is done because we need to train a flat classifier over categories in S.



3. Train the text classifier, using all textual documents under a concept as the training data for that concept.
4. Set up Rainbow to receive classification requests on a specific server port.

DISCUSSION of APPROACH I:
Since we use all documents for training purpose, the number of features per category is very large. A large feature set leads to a performance loss in many cases. Moreover, since different classes have different amounts of textual data under them, classes with larger amounts of textual data appear much more often in classification results as compared to the ones with smaller amounts of textual data. This is because the classification results are based on word probabilities and occurrence counts, which creates a bias towards the classes with more data. This clearly leads to a sub-optimal quality of classification results, and in turn a lower quality of the system generated user profile. To overcome the data imbalance problem, we need a way to 'equalize' the classes and reduce the feature set.

APPROACH II:

Steps 1 and 2 as in APPROACH I.

3. Run Rainbow document classifier at the level 'Top', this time indexing only 20 randomly selected documents under each class. Selecting the same number of documents for each class overcomes the data imbalance problem of APPROACH I.

4. Set up Rainbow as a server on a specific port.

DISCUSSION OF APPROACH II:
In this approach, we intend to reduce the feature set of the TF IDF based classifier. Feature selection in text classification has been repeatedly shown to lead to little accuracy loss, and to a performance gain in many cases. Our method of reducing the features is to select a smaller, fixed number of training documents per category. Selecting a fixed number of training documents per category equalizes the categories, and since each of training documents contain rich textual information about a number web pages, selecting even a small number of training documents per category results in a rich feature set.

3.4 System Software Architecture

In this section, we describe the software architecture of our personalization system. Figure 2 gives an overview of the system. The system is composed of two parts: i) the server-side part which is implemented on a server, and ii) the client-side part which resides on user's iPhone.

3.4.1 Server-Side

The server-side of our system consists of three main components:

1. A text classifier, trained as described in section 3.3
2. A socket program that communicates with the text classifier over a server port.
3. A Django application that receives search query from the user, retrieves Yahoo search results for the query, forwards them for classification and returns the search results along with their classification back to the client device.

Django Application

Django is an open source web application framework, written in Python. The 'Django App' component of Figure 2 is an integral server-side component of our system. It integrates with the Yahoo BOSS search framework. Specifically, the Django App receives search query from the client device and retrieves Yahoo search results for the query using the "BOSS Mashup Framework". It then sends the search results to the 'Socket Program' component, and receives the classification results back from the 'Socket program' component. Finally, it sends the results back to the user.

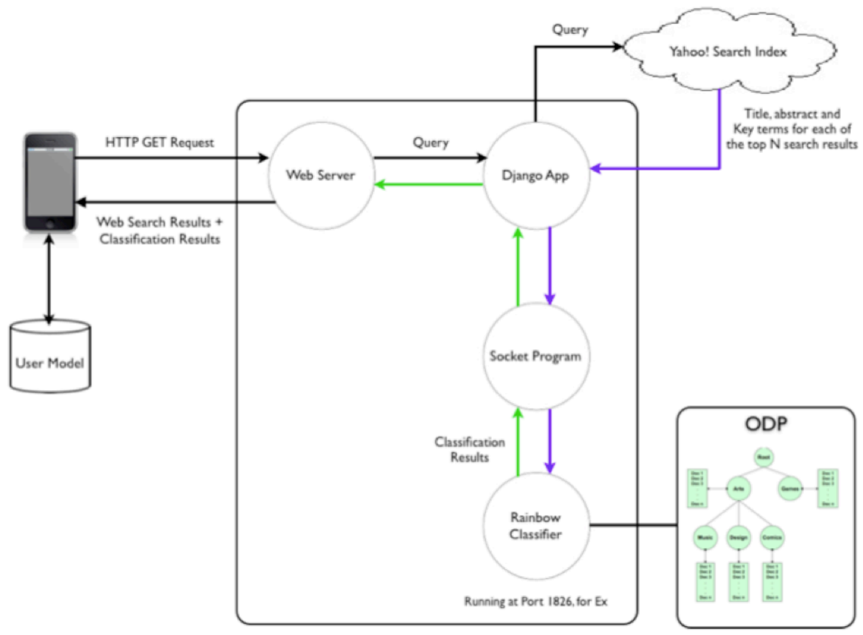


Fig 2: System Architecture

Socket Program
The Socket Program component in Figure 2 is a C program that performs socket communication. We set our Rainbow classifier to be run as a continuous background server process. The classifier listens for document classification requests over a port and produces a classification score for each category for which it was trained. The socket program does the job of sending document classification requests to the port on which Rainbow is running, and reading the document classification result scores back from the port.

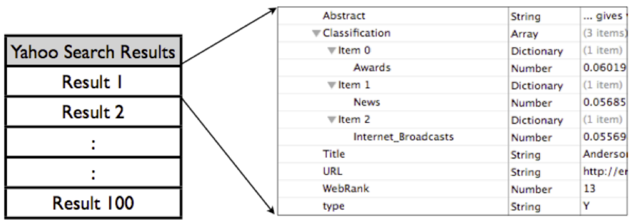
Whenever the user performs a search on his iPhone, an HTTP GET request containing the user query is sent to our web server. The web server is configured to forward such requests to the Django application. The Django application receives the HTTP request URL from the web server and extracts the query from the URL. It then performs Yahoo web search for the query through the BOSS API. We fetch the top 100 search results from Yahoo. That corresponds to the first 10 pages of search results. Given that users typically browse up to the top 2 to 3 result pages on an average, we believe that 100 search results will be reasonable in most cases. The search results are returned as a JavaScript Object Notation (JSON) formatted string. JSON is a lightweight data-interchange format that is based on a subset of the JavaScript Programming Language [10]. For each search result, the JSON string contains the Title, URL, abstract and key terms (among other data) corresponding to the web page. In our Django application, we combine the title, abstract and key terms for each search result into a single string. We believe that the combination of title, abstract and key terms for a web page provides sufficient information of what the web page is about. A more sophisticated approach would be to extract the complete text of the search result web page and analyze it to understand what the web page is about but that requires additional steps such as parsing out all the HTML content and performing text analysis, both of which add significantly to the post-processing time. Besides, the key terms were extracted by Yahoo by performing text analysis in the first place and provide much valuable information about a search result web page in addition to its abstract. Therefore, using the Yahoo key terms is the same as performing text analysis on the web page content. In [9], they take a similar approach as ours wherein they used the Google SOAP API to access Google search results and used the search snippets as representing the search result web page.

Concept	Weight
C_1	W_1
C_2	W_2
:	:
:	:
:	:
C_n	W_n

Table 1: User Profile on Client Side

3.4.2 Client-Side
As discussed earlier, we do not store any type of user information on the server. The user's interest profile is maintained locally on the user's iPhone. We model user interests using the same concepts that we trained our text classifier on.

Table 1 shows the structure of the user profile. The first column contains concept names and the second column contains concept weights in the user model. Initially, all concept weights are zero. The concept weights are constantly updated by our system based on the user's interaction with the search results and based on the links the user visits after clicking one of the search results. At any time, the concepts with higher weights are the ones the user is more likely to be interested in. Figure 3 below shows what information about each search result is returned from the server.



Abstract	String	... gives
▼Classification	Array	(3 items)
▼Item 0	Dictionary	(1 item)
Awards	Number	0.06019
▼Item 1	Dictionary	(1 item)
News	Number	0.05685
▼Item 2	Dictionary	(1 item)
Internet_Broadcasts	Number	0.05569
Title	String	Andersoı
URL	String	http://er
WebRank	Number	13
type	String	Y

Fig 3: Search Result Details Stored on User's iPhone

For each search result, we return the Title, URL, Abstract, Web Rank and the top three categories assigned to that result by our document classifier. Once the Yahoo search results are received, the next step on the client-side is to re- rank the results so that the ones that are more likely to be of interest to the user are shown above others.

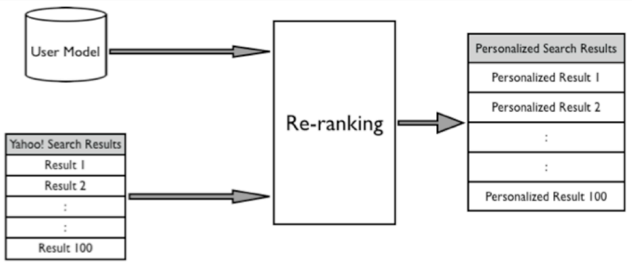


Fig 4: Re-ranking Search Results on the Client Side

The re-ranking is achieved through a matching function, which calculates the degree of similarity between each search result and the user profile.

$$Sim(user_i, Result_j) = \sum_{k=1}^{N} wp_{i,k} \cdot wd_{j,k}$$

where

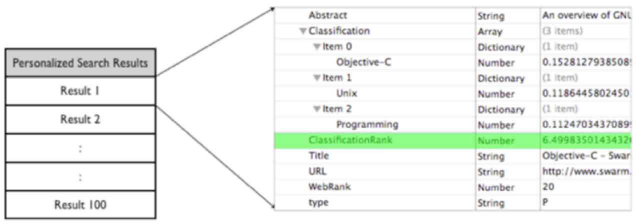
$wp_{i,k}$ = weight of concept k in user profile,
$wd_{j,k}$ = weight of the concept k in the result j,
N = number of concepts returned to the client.

The final weight of the document used for reordering is calculated by combining the previous degree of similarity with Yahoo's original rank, using the following weighting scheme:

$$match(user_i, Result_j) = \alpha \cdot sim(user_i, Result_j) + (1 - \alpha) \cdot YahooRank(Result_j)$$

where α gets values between 0 and 1. When α is 0, conceptual rank is not given any weight, and the match is equivalent to the original rank assigned by Yahoo. If α has a value of 1, the search engine ranking is ignored and pure conceptual match is considered. Obviously, the conceptual and search engine-based rankings can be blended in different proportions by varying the value of α.

The final score of each search result is assigned to the search result as shown in Figure 5. Finally, the search results are sorted based on their final scores, so that the ones with higher scores are ranked higher.



Abstract	String	An overview of GNU
▼Classification	Array	(3 items)
▼Item 0	Dictionary	(1 item)
Objective-C	Number	0.152812793850
▼Item 1	Dictionary	(1 item)
Unix	Number	0.1186445802450
▼Item 2	Dictionary	(1 item)
Programming	Number	0.112470343708
ClassificationRank	Number	6.49983501434320
Title	String	Objective-C - Swar
URL	String	http://www.swarm.
WebRank	Number	20
type	String	P

Fig 5: Post Re-ranking, and Search Result Details Stored on the iPhone

Once the user is presented the re-ranked search results, the system enters into observation mode. Whenever the user clicks a search result, the categories that were assigned to that search result by the classifier are updated. For instance, if categories C1, C2 and C3 were assigned to the search result by the server, C1 being the best match and C2 and C3 being the second and the third best matches, the system would increase the weight of C1 by 3, C2 by 2 and C3 by 1 in the user model. After the user clicks a search result and is viewing a web page, we also monitor the hyperlinks that the user visits from the web page. When the user clicks on a hyperlink, we extract the text from all the paragraph elements of the target webpage. The extracted text is sent to the server for classification. Once the classification results are returned from the server, the top three category matches are updated proportionally in the user model.

4. EXPERIMENTS

To evaluate the effectiveness of our personalized search results, we built an evaluation version of our client-side system. The evaluation version was designed to run on the 'iPhone Simulator' software that is part of the iPhone SDK. In the evaluation version, we combine the top 10 web search results from Yahoo and the top 10 personalized search results. If there is overlap (e.g., when some of the top 10 personalized search results come from the top 10 Yahoo search results), we add an equal number of personalized and Yahoo search results so that the final count of search results displayed to the user is 20. The search results are shuffled before they are displayed to the user so as to remove any bias. Upon clicking a search result we record the fact that the user considered the selected search result relevant. We communicate this to the user by displaying a small tick mark next to the visited search results. If in fact the user thinks otherwise, he can uncheck the search result and the record for that search result will be removed. We asked 5 graduate students (3 from Computer Science, 1 from Textile Science and 1 from Bio Technology) from University of Georgia to use the evaluation version of our app over a period of 7 days. In the rest of the discussion, we refer them as User 1, User 2, etc. The users were first given an overview of our system and were explained the experimental setup (describe above). They were asked to use our application for performing web search just as they would normally query a search engine. Before clicking on any search result for a given query, the users were asked to carefully review the title, abstracts and URLs of all search results and then click on the ones they thought were relevant to them.

Experiment 1: System Generated User Profile vs. True User Profile

Given that our primary goal is to learn a model of user interests based on his interaction with search results, and use this model to personalize search ranking, one natural way to evaluate our learning method is to measure the difference between the user's actual interest vector and the learned interest vector. At the end of the 10 day period of user evaluation, the users were shown the top 20 system predicted user interests were asked to re-order the interests based on what they thought were their true interests. To measure the degree of agreement between the two lists, we calculate normalized Kendall tau distance (see [11], for how normalized Kendall tau distance is calculated) between them. The normalized Kendall tau distance lies in the interval [0, 1], where 0 indicates that the two lists are identical and 1 indicated maximum disagreement.

	Normalized Kendall Tau Distance
User 1	0.19
User 2	0.1
User 3	0.15
User 4	0.27
User 5	0.2

Table 2: Normalized Kendall Tau Distance between the System Predicted Interest Vector and the True Interest Vector

Table 2 shows the normalized Kendall Tau distance value for the five users. We note that the value for all users are closer to 0, which indicates agreement between the system generated interest vector and the true user interest vector. We can therefore assert that our learning method does a good job of identifying user interests.

Experiment 2: Comparing User Interaction with Standard and Personalized Results

In this experiment, we wished to determine which search results the users tended to view more often - personalized search results or the standard search results. For each query, we recorded which search results the user considered relevant. The search results were tagged as 'P' if they came from the personalized results and 'Y' if they came from standard Yahoo search results and 'YP' if they were common to both, the top 10 Yahoo search results and the top 10 personalized results. At the end of the evaluation, we calculated the total number of search results clicked by each user and how many of them were personalized results.

Table 3 compares the percentage of standard and personalized search results clicked by users. It is clear that the users considered the personalized results more relevant compared to the standard search results. And since the experiment presented search results in an unbiased manner, we can assert that the personalized search results were indeed relevant to user needs and that integrating user interests can help improve the quality of web search.

6. CONCLUSION

This research was about personalizing web search on mobile devices. As a case study, we used Apple's iPhone as the client mobile device. Our approach involved building an interest profile on the user's iPhone based on his interaction with web search results and his browsing behavior. Personalization of search results was achieved by re- ranking search results returned by a standard web search engine (Yahoo) based on proximity to the user's interest profile. The ability to recognize user interests in a completely non-invasive way and the accuracy of the personalized results are some of the major advantages of our approach. The average response time of our system for displaying the top 100 personalized search results was found to be less than 2 seconds which is reasonable in a mobile environment. Our experimentation showed that, when presented with an unbiased, randomized list of standard web search results and personalized search results, users viewed personalized results more often than standard web search results. We can therefore assert that search personalization can not only be achieved but can be effective in the mobile environment.

	# Search Results Clicked	# Personalized Results Clicked	# Yahoo Results Clicked	% of Personalized Results Clicked
User 1	171	103	88	60.23%
User 2	229	127	102	55.45%
User 3	226	135	91	59.73%
User 4	160	84	76	52.50%
User 5	174	112	62	64.36%

Table 3: Statistics of the search results clicked

REFERENCES

[1] Jansen, B J, Spink, A, Bateman, J, and Saracevic, T, 1998. Real life information retrieval : A study of user queries on the Web. ACM SIGIR Forum 32, 1, 5 –17

[2] The Librarian's Internet Index - http://lii.org

[3] The Internet Public Library - http://ipl.org

[4] Yahoo Web Directory - http://dir.yahoo.com

[5] Open Directory Project - http://dmoz.org

[6] Evgeniy Gabrilovich, Shaul Markovitch: Harnessing the Expertise of 70,000 Human Editors: Knowledge-Based Feature Generation for Text Categorization. Journal of Machine Learning Research 8 (2007) 2297-2345

[7] http://www.cs.cmu.edu/~mccallum/bow/rainbow/

[8] http://developer.yahoo.com/search/boss/mashup.html

[9] Mirco Speretta, Personalizing Search Based on User Search Histories, Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence (2005) 622 - 628

[10] JSON-http://www.json.org/

[11] Calculating Kendall Tau Distance http://en. wikipedia.org/wiki/Kendall_tau_distance