User profiling using implicit feedback

Anton Lu, Björn Lindqvist, Chun Hung Lin, Sergio Liberman Bronfman

Improving search quality by learning user embeddings using implicit feedback

May 16, 2019

Abstract

In Information Retrieval (IR), user profiling can significantly improve search results by taking user profiles into account. If a user has revealed some information about him or herself, this information can be used to tailor results to his or her needs. However, uses are often reluctant to submit explicit feedback so instead implicit feedback, such as clicks on search results, can be used to build profiles.

We explore several different methods for building and using such implicit feedback to improve relevancy. We propose and evaluate a novel combination of two techniques; query reordering, query expansion, and decaying profiling. The first, query reordering, works by reordering search results retrieved from the underlying search engine according to the user profile. The second, query expansion, by adding terms from the user profile to the query. We also propose a scheme for combining profiles into static and dynamic component, where the latter component is decayed over time so that new information from the user takes precedence over old.

Although we haven't evaluated our implementation on a group of test subjects, the results we present show that using clicks as implicit feedback can work in practice and that our methods improve search quality.

I. Introduction

Personalized Information Retrieval (PIR) is a subfield of Information Retrieval where the user's *profile* influences the retrieval system's behaviour. Two users may have completely different information needs even though their queries are identical. For example, the query "Java" might express an interest in Java the programming language, coffee or the Indonesian island Java. To resolve such ambiguities, IR systems can collect information from users to produce query results tailored to their predicted needs.

User information in the form of explicit feedback, for example Rocchio's algorithm for relevancy feedback, have been shown to greatly enhance search quality [1]. However, users often do not want to spend the time providing explicit feedback so instead, implicit means of gathering feedback have been developed [2]. For example, by keeping track of which search results users clicks on, user profiles can be learned to provide better results in the future. An overview of some methods for implicit user profile construction are given in section II.

The rest of this article is structured as follows. In section III, we describe our proposed system for dynamically updating user embeddings by means of collecting click data and in section IV how we have implemented it. Then in section V we describe how we have evaluated our implementation. An evaluation that unfortunately is incomplete due to our lack of proper test subjects. In section VI we declare our results and in section VII, we discuss them.

II. Background

There are a multitude of methods for customizing the user's search experience based on feedback. We describe a few that have been proposed and found to be useful and also the metric we have used for evaluating our method.

i. Personalized PageRank

The use of personalized PageRank was first suggested in Page, Brin, Motwani and Winograd's landmark paper *The PageRank Citation Ranking:*

Bringing Order to the Web in 1998 [3] but never fully explored by the authors. The method works by adjusting the random surfer model so that the surfer isn't equally likely to jump to a random page but instead is biased towards some pages, such as those found in the user's bookmarks. In this way, each user interacting with the search engine could have his or her own set of unique PageRanks for each web page.

Haveliwala [4] proposed using *Topic-Sensitive PageRank* so that instead of one global PageRank value, each page would be assigned a PageRank for each topic in the system. He used 16 representative topics taken from the Open Directory project in his implementation. Two scenarios were considered; one in which the user enters the query in a text box in which case the PageRank vectors related to the query's topics would the used for ranking and one in which the user searches by selecting text in a document, causing the selection of PageRank vectors to depend on the document's topic.

A problem with Haveliwala's approach is that the system must be able to accurately predict what topic a query belongs to. Qiu and Cho [5] tried to plug this hole by proposing an automatic method for learning user interests which could guide the selection of PageRank vectors. They modelled user preferences using topic vectors and tried to learn these by analyzing the user's click history on the search engine's result pages.

Recent research have incorporated Monte Carlo methods for fast incremental PageRank [6] or by using distributed algorithms [7], making personalized PageRank much more computationally feasible.

ii. Session-based user profiling

Shen, Tan, and Zhai [8] proposed categorizing user preferences as either ephemeral or long-lasting. Ephemeral would be those expressing a users immediate needs, while long-lasting ones would model general user attributes such as education level, general interests and accumulated click-through history. Focusing on ephemeral preferences, they developed a Bayesian model in which each query result would be affected by the previous ones in the same session updated by the user's click-through. In this way, they argued, given a predecessor query of "CGI programming", the query "java" is very likely to be about Java programming.

A similar session-based approach was explored by Daoud et al. [9]. They also discussed a mechanism for automatically discovering session boundaries using the Kendall rank correlation measure.

Sugiyama, Hatano, and Yoshikawa [2] introduced an expontential decaying factor to emulate that the perference of a user steady changes as time pass.

iii. User embeddings

Several authors have explored user embeddings as a means to improve search. That is, representing user preferences as vectors and having their similarities with document vectors influence document rankings.

TF-IDF vector embeddings

Matthijs and Radlinski [10] extended the the standard TF-IDF vector model by embedding user preferences in the same vector space and ranking documents by their similarity to the query vector and the user vector. Users were represented by lists of terms and weights learned from users' browsing histories and clicks.

Xu et al. [11] proposed a similar approach. But instead of using browser history and clicks, vectors of user's interests were learned by observing what tags they were using on a social media platform where the experiment were conducted.

Semantic embeddings

Several authors have tried to model user preferences using sets of general topics with associated weights. Vu et al. [12] called their approach "embeddings using topic vectors." Optimal user vectors were learned using stochastic gradient descent on a set of positive and negative training examples consisting of document/query pairs subjectively determined to be relevant or irrelevant. The topic vectors of the documents in the collection were first learned using Latent Dirichlet Allocation.

Ai et al. [13] likewise proposed a user embedding model based on semantic representations and showed it to perform very well for personalized product search. They used hierarchial embeddings and optimized the model using deep learning.

iv. Evaluation metric

The primary evaluation metric we have used in this work is the *Normalized Discounted Cumulative Gain* (nDCG) at position p [14], defined as

$$\mathrm{nDCG_p} = \frac{\mathrm{DCG_p}}{\mathrm{IDCG_p}},$$

where $\mathrm{DCG_p}$ is further defined as

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{2^{\mathrm{rel_i}} - 1}{\log_2(i+1)}$$

and $\mathrm{IDCG_p}$ (Idealized Discounted Cumulative Gain at position p) is defined as

$$\mathrm{IDCG_p} = \sum_{i=1}^{|REL|} \frac{2^{\mathrm{rel_i}} - 1}{\log_2(i+1)},$$

where $|REL|$ represents a list of the top p most relevant documents in the corpus ordered by relevance. rel_i is an integer from 0 to 3 – the subjective relevance of the document ranked at position i for the user.

The metric produces a number from 0 to 1, indicating how much of the theoretical "maximum search goodness" is captured. For example, $\mathrm{nDCG_p} = 0.7$ could be thought of capturing 70% of the first p search results "search goodness".

In our experiments we let p be 20, which is the number of hits Google's search engine displays on its first two result pages. As others have shown, users are unlikely to investigate more results than those many [15].

III. Methods

We now describe our proposed mechanisms for exploiting implicit feedback to build user profiles to improve search quality. First, we discuss how we represent users and documents and then how we use the information for query expansion and re-ranking of search results.

i. User and document representation

Users are represented using bags of words; lists of terms and weights associated with them. Terms and weights were collected from three sources of varying weights:

- *Document titles*: terms of titles of documents clicked on.
- *Document categories*: terms of categories of documents clicked on.
- *Document content*: terms of the contents of documents clicked on.

Therefore, for each user we store three vectors, $\mathbf{u_t}$, $\mathbf{u_c}$, and $\mathbf{u_x}$, containing weights of terms collected from these fields.

Similarily, for every document d_i we associate three vectors, $\mathbf{t_i}$, $\mathbf{c_i}$, and $\mathbf{x_i}$, for terms from the fields mentioned above.

To the three sources, we associate three weights, w_t, w_c, and w_x , with which we compute user and document vectors as weighted sums,

$$\mathbf{u} = w_t\mathbf{u_t} + w_c\mathbf{u_c} + w_x\mathbf{u_x}$$
$$\mathbf{d_i} = w_t\mathbf{t_i} + w_c\mathbf{c_i} + w_x\mathbf{x_i}.$$

In all calculations, these vectors are normalized to unit length. For both users and documents, we use the L_2 norm and TF-IDF vectors.

ii. Query expansion

Query expansion is a method for reformulating search queries in order to improve results. Reformulating queries based on the user profiles specified in the previous section can improve results specific for that user. Our implementation is loosely based on the Rocchio formulation and the user profile vector is considered as a document.

The input query term vector $\mathbf{q}$ is therefore expanded as follows

$$\mathbf{q}_{new} = \mathbf{q} + \gamma\tilde{\mathbf{u}},$$

where $\gamma \in [0..1]$ is a parameter controlling the weight of the modified user vector, $\tilde{\mathbf{u}}$. To construct $\tilde{\mathbf{u}}$, we pick only the top 100 scoring terms of the user profile, excluding those already specified by the query.

We limit the number of terms we pick from the user profile to improve performance because the more query terms the slower the query becomes. And terms with tiny weights doesn't impact the search results much anyway. 100 terms seemed like a good tradeoff between strong personalization and search performance.

We also exclude terms already in the query to avoid double counting; if the same terms are in both the original query and the modified profile, those terms would be overemphasized and decrease the impact of the rest of the profile's terms.

The modified profile vector is also normalized before it is added to the query.

iii. Re-ranking strategy

Re-ranking is performed by fetching a sufficiently large number of documents from the underlying search engine and then reassigning their relevance scores using the user profile.

For each document d_i, let $\mathbf{d_i}$ be its weighted term vector sum. Furthermore let s_i be the score assigned to it by the search engine. Then its final re-ranked score is computed as

$$r_i = \alpha s_i + (1 - \alpha)\mathbf{u}^T\mathbf{d_i},$$

where $\alpha \in [0..1]$ is a parameter controlling how much weight is given to the user profile.

iv. Adaptive profile updating

To model changes in user preferences, we employ the expontential decaying scheme introduced by Sugiyama, Hatano, and Yoshikawa [2]. This way, a user preference expressed a long time ago carries less weight than one expressed recently. We therefore model the user profile vectors as functions of time, represented using

$$\mathbf{u_k}(t) = \mathbf{u_k}(t') \exp(-\lambda_k(t - t')),$$

where $\mathbf{u_k}$ is one of the three user vectors; title $\mathbf{u_t}$, category $\mathbf{u_c}$, or content $\mathbf{u_x}$ and λ_k is the associated exponential decay constant. It is in turn defined as a function of the half-life value $t_{\frac{1}{2},k}$ as follows

$$\lambda_k = \frac{\ln 2}{t_{\frac{1}{2},k}}.$$

This means that different decay constants apply for different profile vectors, rather than setting them all the same. This is in constrast to the scheme proposed by Sugiyama, Hatano, and Yoshikawa [2] in which only one constant was used with the value 0.099 day^{-1}, meaning that the user preferences decreased by half after one week. We instead claim that pages' titles and categories are more important and should last longer than the content of pages.

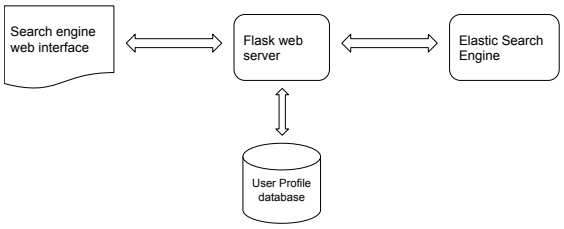

Figure 1: The system diagram of the implementation

When a user clicks on a document in the search results at time t_0, the vectors from that document are added to the user's profile according to the following scheme:

$$\mathbf{u_k}(t_0) = \mathbf{u_k}(t') \exp(-\lambda_j(t_0 - t')) + \mathbf{u_k}',$$

where t' is the previous update time and $\mathbf{u_k}'$ is the profile vector to be added into the existing profile vector.

v. Static and dynamic user profile

The user profile consists of a static and a dynamic component. The static component consists of the keywords provided by the user and is never updated. Only the dynamic part is updated when the user clicks on search results. The profile is described by

$$\mathbf{u}(t) = (1 - \beta)\mathbf{u}_s + \beta\mathbf{u}_d(t),$$

where $\beta \in [0..1]$ is a parameter controlling the weight of the dynamic part of the profile.

IV. Implementation and datasets

Our implementation uses ElasticSearch 6.3.1, a free Java search engine based on the Lucene library. It contains many useful features such as TF-IDF scoring, stemming and stop word pruning which greatly improves the quality of our rankings.

To present search results to users, we have built a website using Materialize and Vue.js with a Python Flask backend. We capture users' clicks on search

User	Query	Information need
A	java	Java the Indonesian island
B	speer	The Nazi Albert Speer
C	gamla stan	Metro station Gamla stan
D	fredrik och filip	The television presenters
E	java	Java programming

Table 1: Queries and information needs for our personas.

results using JavaScript and forward the information back to the backend for updating the user profile. User profiles are stored in a SQLite database.

For each query issued, we fetch the first 50 results from ElasticSearch, with or without query expansion. Then we reorder them according to the algorithm described in section III and finally we display them to the user.

i. Search corpus

The dataset we use for evaluation is a dump of the Swedish Wikipedia customized for ElasticSearch, generated on April 29, 2019.[1] It contains a total of 11,548,339 documents.

ii. Subjective ground truth

Since we didn't have access to a test group to run our experiments on, we instead developed a set of six fictional "personas" on which to conduct experiments. For each persona, we imagined an information need that likely would be aided by user profiling. For example, someone from the Indonesian island Java searching for "Java" probably wants information about the island and not the programming language. Our five personas, their queries, and information needs are listed in table 1.

To evaluate if our system helps such users, we annotated documents retrieved from the system for the queries on an integer scale from 0 to 3, using the relevance criteria defined in Sormunen [16]. There was an unfortunate, but unavoidable bias involved in this process as we had to imagine ourselves as if we were in the users shoes.

[1] Available from `https://dumps.wikimedia.org/other/cirrussearch/20190429/`

V. Evaluation

To evaluate the effectiveness of our profiling system, we ran two experiments described in the following sections.

i. Experiment 1

In the first experiment, we first ran the test queries in table 1 without profiling and computed their $nDCG_{20}$. We then simulated the profiles being updated by clicking on ten relevant and irrelevant documents and observing what effect that had on the respective search queries. The order of each user's clicks and subjective relevance ratings are documented in section VIII.

Each simulated user started out with an empty profile and the 100 terms with the highest weight were used to expand the search query. To make the results easier to reproduce, no adaptive profile updating were used. Both users and documents were represented using TF-IDF vectors. The results were then reranked using $\alpha = 0.7$.

To test the relative impact of the query expansion feature versus reordering, we also ran the same test with expansion disabled. In this variant we used $\alpha = 0$ and the 500 top terms of the profiles were used to reorder query results.

ii. Experiment 2

The second experiment is to evaluate the impact of the weigthing (β) between static and dynamic profiles. We considered a scenario that a user is going to Japan for sightseeing and she/he needs the search engine to obtain the information related to sightseeing. We assigned the static profile for a user as shown in table 2 and the user dynamic profile was built through searching and viewing Japan related pages like Japan cities and Japanese food, culture, and architecture. The dynamic profile after building it is shown in table 3.

We then tried three queries to evaluate the search result quality with different β values. The queries were "flygplats", "slott", and "tempel" which mean airport, castle, and temple in English respectively. Four β values were tested in this experiment and they were 0.9, 0.7, 0.3 and 0.1.

Term	Weight
amerika	0.447
matematik	0.447
anime	0.447
programming	0.447
kina	0.447

Table 2: The static profile vector for the user in the experiement 2

Term	Weight
japan	0.7727
japansk	0.3195
wp:projek	0.2843
text	0.1813
kyoto	0.1709
prefektur	0.1707
osak	0.1371
kall	0.1264
behov	0.1242
tempel	0.1122

Table 3: The dynamic profile vector for the user in the experiement 2

We scored the search result as 3 when it was in, related to, or the surrounding of Japan. For example, Kyoto were scored as 3 for searching temple because Kyoto has a lot of Japanese temples and Shinto shrines. We scored the search result as 2 when it was purely exactly matched the query itself and scored the search result as 1 when it was outside Japan. We scored 0 when the result was completely irrelvant.

VI. Results

Results of our first experiment are presented in figure 2, figure 3, figure 4 and figure 5. The nDCG figures shows how the $nDCG_{20}$ changes as more and more documents are clicked on. Query expansion is on in the first figure and off in the second one.

For most personas the metric increases the more clicks the user submits in both variants. This indicates that user profiling could help users. It is

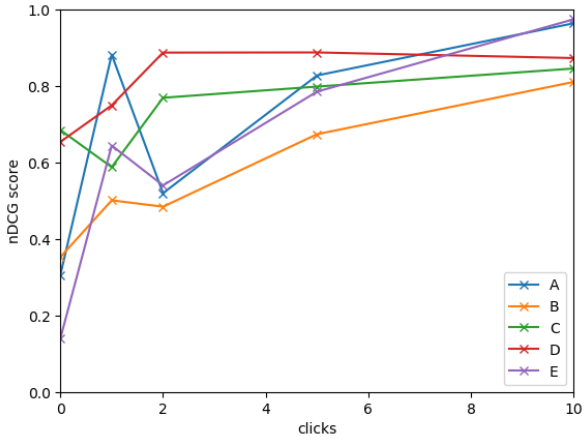

Figure 2: $nDCG_{20}$ metric for the five users queries as they click on documents. Results includes query expansion and reordering.

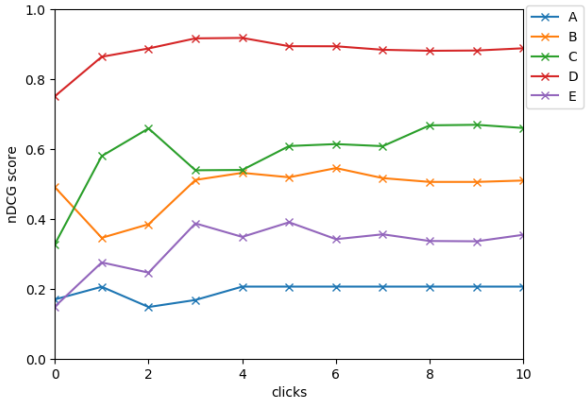

Figure 3: $nDCG_{20}$ metric for the five users queries as they click on documents. Results includes only reordering.

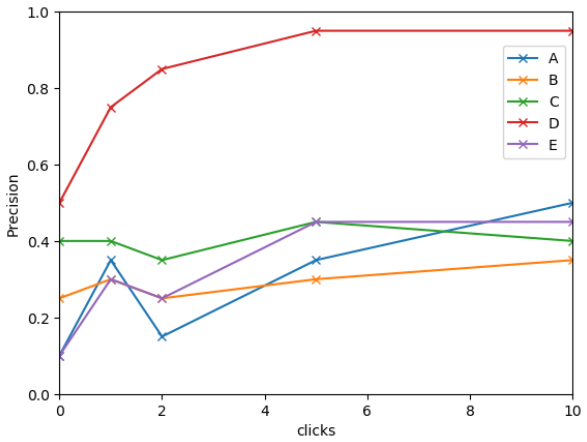

Figure 4: Precision for the first 20 search results using both query expansion and reordering.

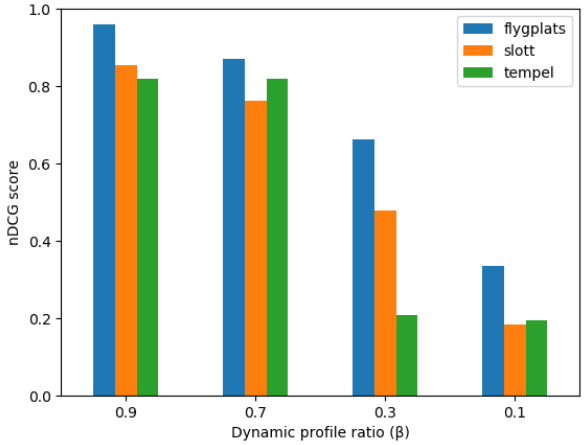

Figure 6: $nDCG_{10}$ metric for the four dynamic profile ratio β and three queries "flygplats", "slott", and "tempel"

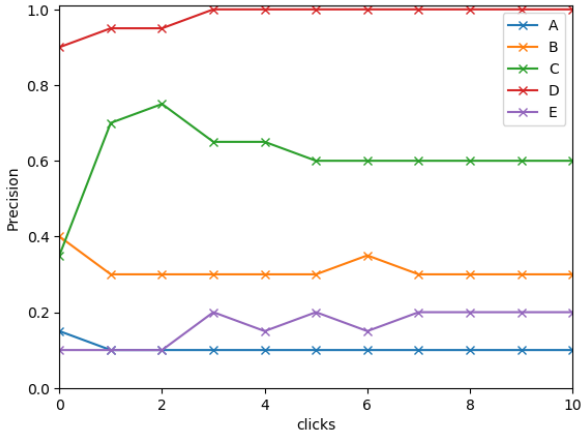

Figure 5: Precision for the first 20 search results using only reordering.

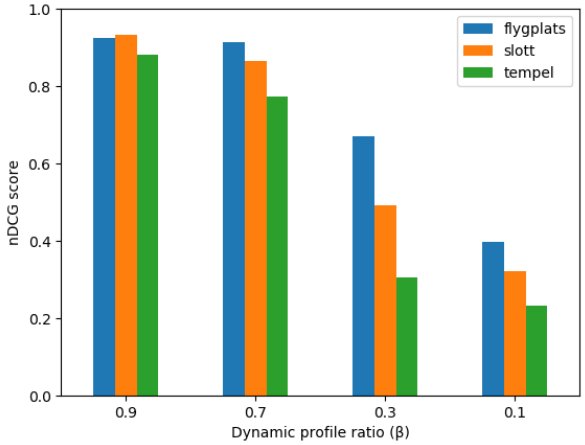

Figure 7: $nDCG_{20}$ metric for the four dynamic profile ratio β and three queries "flygplats", "slott", and "tempel"

Figure 8: A word cloud visualization of user A's profile after 10 search results have been clicked on.

Term	Weight
jav	0.8731
majapahitriket	0.2114
sydostasiatisk	0.1943
histori	0.1593
stat	0.1451
indonesi	0.1426
historisk	0.1336
singhasari	0.1073
enwp	0.1056
sundariket	0.1007

Table 4: The truncated profile vector of the persona User A after the experiement

interesting to note that the search quality increases even without query expansion, albeit not as much. Relying only on reordering can still be useful as it does not impact the ranking of unrelated queries as much.

Results of our second experiment are presented in figure 6 and figure 7. The result shows that when the portion of the dynamic profiling part becomes larger, the search engine can capture the user recently behaviour better and provide better suggestions which fit the user's temporary needs.

VII. Discussion

We have implemented and evaluated a novel approach combining a static user profile with one built using the users search and click history. The main strength of this approach is that it is very flexible and there are many ways in which the behaviour of the system can be adjusted. For example, we can tweak the weigthings for the dynamic profile, the weigthings between different fields of the docuement, and the half-lifes of the fields.

A weakness of our approach is that we only have positive feedback – clicks are always considered positive and never negative. But, for example, if a user clicks on a page but shortly thereafter leaves it that can be seen as a negative signal. Also, there is no mechanism to "upgrade" dynamic profiles to static ones. The persistent component of the dynamic profile would relatively decrease the importance of the rest of the dynamic profile components.

The results we have obtained indicate that learning user embeddings using implicit feedback can both be applied for query expansion and result reordering. Both methods improve search quality. We also note that, for each query, most of the gain occur after the first few clicks. Our conclusion is that only a little implicit feedback is enough to personalize queries and improve search quality.

However, we did not have access to a testing group so we can't draw to many general conclusions of our experiment. Also our experiments were subjective because we ourselves created relevancy rankings and simulated what we thought would be realistic user clicks. Our queries we also handpicked and likely works better with profiling than average user queries. Perhaps we would have gotten worse results had we used a more representative set of queries.

One interesting question is when user profiling is useful and when it is not. Searches on Wikipedia, which is the corpus we have used, are perhaps not suitable for personalization. Perhaps for Wikipedia you already know what you want and personalization isn't of much use? Personalization likely helps other kinds of searches more, such as searching in online phone books because you are likely to be searching for a friend or relatives phone number.

References

[1] Mark Van Uden. "Rocchio: Relevance feedback in learning classification algorithms". In: *Proceedings of the ACM SIGIR Conference*. 1998.

[2] Kazunari Sugiyama, Kenji Hatano, and Masatoshi Yoshikawa. "Adaptive Web Search Based on User Profile Constructed Without Any Effort from Users". In: *Proceedings of the 13th International Conference on World Wide Web*. WWW '04. New York, NY, USA: ACM, 2004, pp. 675–684.

[3] Larry Page et al. *The PageRank Citation Ranking: Bringing Order to the Web*. 1998.

[4] Taher H. Haveliwala. "Topic-sensitive PageRank". In: *Proceedings of the 11th International Conference on World Wide Web*. WWW '02. Honolulu, Hawaii, USA: ACM, 2002, pp. 517–526.

[5] Feng Qiu and Junghoo Cho. "Automatic Identification of User Interest for Personalized Search". In: *Proceedings of the 15th International Conference on World Wide Web*. WWW '06. Edinburgh, Scotland: ACM, 2006, pp. 727–736.

[6] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. "Fast Incremental and Personalized PageRank over Distributed Main Memory Databases". In: *CoRR* abs/1006.2880 (2010). arXiv: 1006.2880.

[7] Tao Guo et al. "Distributed Algorithms on Exact Personalized PageRank". In: *Proceedings of the 2017 ACM International Conference on Management of Data*. SIGMOD '17. Chicago, Illinois, USA: ACM, 2017, pp. 479–494.

[8] Xuehua Shen, Bin Tan, and ChengXiang Zhai. "Context-sensitive Information Retrieval Using Implicit Feedback". In: *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '05. Salvador, Brazil: ACM, 2005, pp. 43–50.

[9] Mariam Daoud et al. "A Session Based Personalized Search Using an Ontological User Profile". In: *Proceedings of the 2009 ACM Symposium on Applied Computing*. SAC '09. Honolulu, Hawaii: ACM, 2009, pp. 1732–1736.

[10] Nicolaas Matthijs and Filip Radlinski. "Personalizing Web Search Using Long Term Browsing History". In: *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*. WSDM '11. Hong Kong, China: ACM, 2011, pp. 25–34.

[11] Shengliang Xu et al. "Exploring Folksonomy for Personalized Search". In: *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '08. Singapore, Singapore: ACM, 2008, pp. 155–162.

[12] Thanh Vu et al. "Search Personalization with Embeddings". In: *CoRR* abs/1612.03597 (2016). arXiv: 1612.03597.

[13] Qingyao Ai et al. "Learning a Hierarchical Embedding Model for Personalized Product Search". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '17. Shinjuku, Tokyo, Japan: ACM, 2017, pp. 645–654.

[14] Kalervo Järvelin and Jaana Kekäläinen. "Cumulated Gain-based Evaluation of IR Techniques". In: *ACM Trans. Inf. Syst.* 20.4 (Oct. 2002), pp. 422–446.

[15] Muh-Chyun Tang and Ying Sun. "Evaluation of web-based search engines using user-effort measures". In: *Library and Information Science Research Electronic Journal* 13.2 (2003).

[16] Eero Sormunen. "Liberal Relevance Criteria of TREC -: Counting on Negligible Documents?" In: *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '02. Tampere, Finland: ACM, 2002, pp. 324–330.

VIII. Appendix

In this section we specify the relevancy ratings for each user and what their document click order was in our experiment.

i. User A

- **Relevancy 3**: Java
- **Relevancy 2**: Borobudur, Javanesiska, Javatiger, Kungariket Medang, Majapahitriket, Mongolernas invasion av Java, Singhasari, Sundariket, Yogyakarta (region)
- **Relevancy 1**: Java (olika betydelser)

Click order: Java, Javatiger, Mongolernas invasion av Java, Majapahitriket, Sundariket, Singhasari, Java, Java (olika betydelser), Java, Majapahitriket

ii. User B

- **Relevancy 3**: Albert Speer, Albert Speer Jr., Albert Friedrich Speer, Arkitektur i Nazityskland, Spandaufängelset
- **Relevancy 2**: Transportflotte Speer, Rudolf Wolters, Reichsparteitagsgelände, Volkshalle
- **Relevancy 1**: Legion Speer, Posenkonferensen, Nationalsozialistisches Kraftfahrkorps, Gitta Sereny, P-1000, Rikskansliet i Berlin, Fyraårsplanen, Heinrich Tessenow, Siegesallee, Welthauptstadt Germania, Organisation Todt

Click order: Legion Speer, Nazism i Sverige, Albert Speer Jr., Albert Friedrich Speer, Adolf Hitler, Obersalzberg, Spandaufängelset, Albert Speer, Arkitekturskolan, P-1000

iii. User C

- **Relevancy 3**: Gamla stan, Gamla stan (tunnelbanestation)
- **Relevancy 2**: Apoteket Kronan (Gamla stan), Maria-Gamla stans stadsdelsområde, Stadsplanering i Gamla stan,
- **Relevancy 1**: Kanslihuset, Kvarteret Æolus, Kvarteret Ajax, Kvarteret Amphitryon, Kvarteret Argus, Kvarteret Bacchus, Kvarteret Bootes, Kvarteret Cassiopea, Kvarteret Castor, Kvarteret Cephalus, Kvarteret Cepheus, Kvarteret Diana, Kvarteret Erisichton, Kvarteret Eurydice, Kvarteret Galatea, Kvarteret Ganymedes, Kvarteret Iphigenia, Kvarteret Milon, Kvarteret Neptunus, Kvarteret Neptunus större, Kvarteret Pan, Kvarteret Rådstugan, Kvarteret Python, Kvarteret Typhon, Hebbes bro, Mälartorget, Sammanbindningsbanan (järnväg), Lista över kvarter i Gamla stan, Gator och torg i Gamla stan, Lista över offentlig konst i Gamla stan i Stockholm

Click order: Lista över kvarter i Gamla stan, Gamla stan (tunnelbanestation), Munkbrohamnen, Gröna linjen, Stadsplanering i Gamla stan, Akalla, Riksdagen, Gamla riksdagshuset, Sammanbindningsbanan, Gamla stan

iv. User D

- **Relevancy 3**: 100 höjdare (säsong 1), 100 höjdare (säsong 2), 100 höjdare (säsong 6), Alla mot alla, Alla mot alla med Filip och Fredrik, Boston Tea Party (TV-program), Breaking News med Filip och Fredrik, Filip och Fredrik, Filip och Fredriks podcast, Grattis världen, High Chaparall, Jorden runt på 6 steg, La Bamba (TV-program), Lista över avsnitt av Vem kan slå Filip och Fredrik, Lite sällskap, Nexiko, Ska vi göra slut?, Så tar du ut din lön i choklad, Söndagsparty med Filip & Fredrik, Tårtgeneralen, The Filip and Fredrik podcast, Ursäkta röran (vi bygger om), Vem kan slå Filip och Fredrik
- **Relevancy 2**: Ett herrans liv (svenskt TV-program), Filip Hammar, Fredrik Wikingsson, Får vi följa med?, Hello Sydney, Hissen, Kristallen 2012, Nugammalt, Racet till Vita huset, Öppna dagar
- **Relevancy 1**: Anders Adali, Edwin Johnsson, Emil Österholm, Lars Beckung, Stockholm-Köpenhamn, Sverige dansar och ler, Trevligt folk

Click order: Breaking News med Filip och Fredrik, High Chaparall, Kanal 5, Nexiko, Anders Adali, Mässling, Stockholm-Köpenhamn, Jan Myrdal, Nugammalt, Lite sällskap

v. User E

- **Relevancy 3**: Java (programspråk), Java Development Kit,
- **Relevancy 2**: Iterator, Java Desktop System, Java Runtime Environment, Java Virtual Machine
- **Relevancy 1**: ++, D (programspråk), Datorprogram, E (programspråk), Groovy (programspråk), Haxe (programspråk), Interpreterande programspråk, Java Servlet, J Sharp, Multiplattform, Pascal (programspråk), Plain Old Java Object, Programspråk

Click order: Java (programspråk), J Sharp, Java Virtual Machine, Java Development Kit, Haxe (programspråk), Java (programspråk), Java Virtual Machine, Java (olika betydelser), Java Development Kit, Haxe (programspråk)