Recommending Case Bases: Applications in Social Web Search

Zurina Saaya, Barry Smyth, Maurice Coyle, and Peter Briggs

CLARITY: Centre for Sensor Web Technologies,
School of Computer Science and Informatics,
University College Dublin, Ireland
firstname.lastname@ucd.ie
http://www.clarity-centre.org

Abstract. For the main part, when it comes to questions of retrieval, the focus of CBR research has been on the retrieval of cases from a repository of experience knowledge or case base. In this paper we consider a complementary retrieval issue, namely the retrieval of case bases themselves in scenarios where experience may be distributed across multiple case repositories. We motivate this problem with reference to a deployed social web search service called *HeyStaks*, which is based on the availability of multiple repositories of shared search knowledge, known as *staks*, and which is fully integrated into mainstream search engines in order to provide a more collaborative search experience. We describe the case base retrieval problem in the context of HeyStaks, propose a number of case base retrieval strategies, and evaluate them using real-user data from recent deployments.

Keywords: Social search, context recommendation.

1 Introduction

This paper is about social search and the use of case-based reasoning (CBR) techniques to develop social search technologies that work in tandem with mainstream web search engines. Case-based reasoning is well suited to this problem because CBR methods provide us with a framework for reasoning with experiences and, at its heart, social search is about harnessing the search experiences of others in order to improve web search. To this end we will focus on the HeyStaks social search system, which has been described and evaluated in some detail previously [15,16]. Briefly, HeyStaks provides for a range of search engine enhancements to support collaborating searchers, as well as deeper algorithmic components in order to identify relevant search experiences from a community of collaborators. In short, HeyStaks makes result recommendations to searchers at search time, based on the past searches of their social network. It assumes asynchronous, remote collaboration: searchers do not need to be co-located and collaboration can occur overtime as recent searchers benefit from recommendations that originate from earlier search sessions.

A. Ram and N. Wiratunga (Eds.): ICCBR 2011, LNAI 6880, pp. 274–288, 2011.

The HeyStaks recommendation engine borrows many ideas from case-based reasoning work and in this paper we focus on a particular challenge for HeyStaks and its users. Specifically, the central concept in HeyStaks is the notion of a *search stak*, which acts like a folder for our search experiences. Briefly, a user can create a search stak on a topic of their choosing and they can opt to share this stak with other users. As they search (using HeyStaks in combination with their favourite mainstream search engine) the results that they select (or tag or share) will be associated with their active stak. These results can be subsequently recommended to other stak members in the future when appropriate. In this way, stak members can benefit from the past searches of friends or colleagues who share their staks.

Search staks are effectively case bases of search knowledge. As described in [15] each stak is made up of a set of *search cases* that reflect the history of search on a particular page. HeyStaks reuses these cases at search time as a source of recommendations, by suggesting pages that match their queries and that are contained within staks that they have joined or created. In addition, as users locate pages of interest as they search, HeyStaks adds this information to relevant staks and so search experience grows through usage. And thus the relevance to case-based reasoning is that HeyStaks is a multi-case-base CBR system and the stak selection problem outline above amounts to a case base selection problem.

A key problem for HeyStaks is to ensure that the right stak is chosen for a given search session. One way to address this is to ask users to pick their stak at the start of their search session, but since many users forget to do this, this is not a practical solution in reality. The alternative is to use information about the user's current search session as the basis for automatically selecting and/or recommending an appropriate stak at search time, which if successful provides for a much more reliable solution. In this paper then we focus on this stak selection (or case base selection) problem and in what follows we describe and evaluate a recommendation-based strategy that works well enough in practice to automatically suggest relevant staks to the user at search time, or even automatically switch users into a likely stak without their intervention.

2 Related Work

Ultimately this work is focused on the application of case-based reasoning concepts and techniques to support web search. Of course CBR researchers have already recognised the opportunity for case-based techniques to improve information retrieval and web search. For example, the work of Rissland [13] looks at the application of CBR to legal information retrieval, and [4] describe a case-based approach to question-answering tasks. Similarly, in recent years there has been considerable research looking at how CBR techniques can deal with less structured textual cases. This has led to a range of so-called *textual CBR* techniques [10]. In the context of Web search, one particularly relevant piece of work is the *Broadway* recommender system [7], and specifically the Broadway-QR query refinement technique, which uses case-based techniques to reuse past

query refinements in order to recommend new refinements to searchers. The work of [5] applies CBR techniques to Web search in a different way, by combining user profiling and textual case-based reasoning to dynamically filter Web documents according to a user's learned preferences. This paper focuses on how CBR techniques can be applied to conventional Web search, as opposed to related information retrieval tasks. It builds on previous work [1,2,3] which has already demonstrated the benefits of reusing search experiences within community-based search case bases; each case base representing the prior search experiences of a community of like-minded searchers.

An interesting feature of this work is the fundamental role that multiple case bases play in search support. Conventionally, most CBR systems have assumed the availability of a single case base, focusing on issues of case representation, retrieval, adaptation, and learning with respect to this single case base. However, some researchers have considered the potential and challenges for the use of multiple case bases during problem solving. For example the work of [17] introduced the idea of *multi-case-base reasoning* (MCBR) and proposed a novel distributed case-based reasoning architecture to supplement local case base knowledge, by drawing on the case bases of other CBR systems. Building on this concept, [9] explore, in detail, the issues arising out of MCBR, summarizing key component processes, the dimensions along which these processes may differ, and some of the key research issues that must be addressed for successful MCBR systems. The work of [8] goes on to explore varies strategies for implementing MCBR techniques and specifically proposes methods for automatically tuning MCBR systems by selecting effective dispatching criteria and cross-case-base adaptation strategies. The methods require no advance knowledge of the task and domain: they perform tests on an initial set of problems and use the results to select strategies reflecting the characteristics of the local and external case-bases.

In the present paper we are also concerned with a form of multi-case base reasoning. As per the introduction our case base are repositories of search knowledge (search staks), which a particular user has subscribed to, and the specific task that we focus on is the selection of the right case base (stak) for a given query, which of course represents just one of the many processes involved in multi-case-base reasoning. In the case of our work, however, it is a vital process since the lack of an effective case base recommendation technique seriously limits the effectiveness of the HeyStaks system and can lead to a contamination effect across search staks since off-topic content may be added to staks if recommendation failures occur.

3 A Review of HeyStaks

In designing HeyStaks our primary goal has been to provide social Web search enhancements, while at the same time allowing searchers to continue to use their favourite search engine. HeyStaks adds two basic features to any mainstream search engine. First, it allows users to create *search staks*, as a type of folder for their search experiences at search time, and the creator can invite members

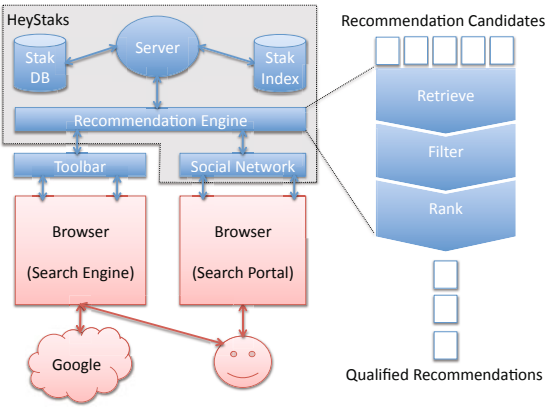

Fig. 1. The HeyStaks system architecture and outline recommendation model

directly. Staks can be configured to be *public* (anyone can join) or *private* (invitation only). Second, HeyStaks uses staks to generate recommendations that are added to the search results that come from the underlying mainstream search engine. These recommendations are results that stak members have previously found to be relevant for similar queries and help the searcher to discover results that friends or colleagues have found to be interesting, results that may otherwise be buried deep within Google's default result-list.

As shown in Figure 1, HeyStaks takes the form of two basic components: a client-side *browser toolbar* and a back-end *server*. The toolbar (see Figure 2) allows users to create and share staks and provides a range of ancillary services, such as the ability to tag or vote for pages. The toolbar also captures search result click-thrus and manages the integration of HeyStaks recommendations with the default result-list. The back-end server manages the individual stak indexes (indexing individual pages against query/tag terms and positive/negative votes), the stak database (stak titles, members, descriptions, status, etc.), the HeyStaks social networking service and, of course, the recommendation engine. In the following sections we review how HeyStaks captures search activities within search staks and how this search knowledge is used to generate and filter result recommendations at search time; more detailed technical details can be found in [16].

3.1 Profiling Stak Pages

Each stak in HeyStaks captures the search activities of its stak members within the stak's context. The basic unit of stak information is a result (URL) and each stak (S) is associated with a set of results, $S = \{r_1, ..., r_k\}$. Each result is effectively a *case* that is anonymously associated with a number of implicit and explicit interest indicators including: the total number of times a result has been selected (Sl), the query terms ($q_1, ..., q_n$) that led to its selection, the terms

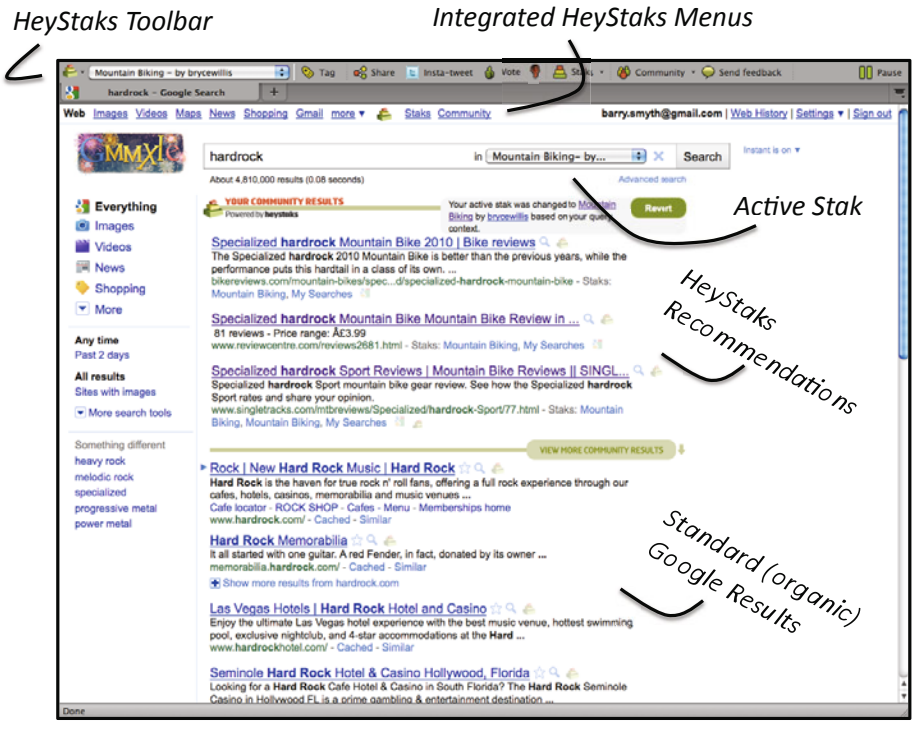

Fig. 2. The searcher is looking for information from a specialist mountain biking brand, Hard Rock, but Google responds with results related to the restaurant/hotel chain. HeyStaks recognises the query as relevant to the the searcher's*Mountain Biking* stak and presents a set of more relevant results drawn from this stak.

contained in the snippet of the selected result $(s_1, ..., s_j)$, the number of times a result has been tagged (Tg), the terms used to tag it $(t_1, ..., t_m)$, the votes it has received (v^+, v^-), and the number of people it has been shared with (Sh) as indicated by Equation 1.

$$r_i^S = \{q_1...q_n, s_1...s_j, t_1...t_m, v^+, v^-, Sl, Tg, Sh\} \; . \tag{1}$$

Thus, each result page is associated with a set of *term data* (query terms and/or tag terms) and a set of *usage data* (the selection, tag, share, and voting count). The term data is represented as a Lucene (*lucene.apache.org*) index, with each result indexed under its associated query and tag terms, and this provides the basis for retrieving and ranking *recommendation candidates*. The usage data provides an additional source of evidence that can be used to filter results and to generate a final set of recommendations.

3.2 Recommending Results: Relevance and Reputation

At search time, the searcher's query q_T and current, active stak S_T are used to generate a list of recommendations to be returned to the searcher. A set of *recommendation candidates* are retrieved from S_T by querying the corresponding Lucene index with q_T. This effectively produces a list of recommendations based on the overlap between the query terms and the terms used to index each recommendation (query, snippet, and tag terms). These recommendations are then filtered and ranked. Results that do not exceed certain activity thresholds are eliminated as candidates; e.g., results with only a single selection or results with more negative votes than positive votes (see [16]). The remaining recommendation candidates are then ranked according to two key factors: *relevance* and *reputation*. Essentially each result is evaluated using a weighted score of its relevance and reputation score as per Equation 2; where w is used to adjust the relative influence of relevance and reputation and is usually set to 0.5.

$$score(r, q_T) = w \times rep(r) + (1 - w) \times rel(q_T, r) \ . \tag{2}$$

The relevance of a result r with respect to a query q_T is computed based on Lucene's standard *TF*IDF* metric [6] as per Equation 2. The reputation of a result is a function of the reputation of the stak members who have added the result to the stak. And their reputation in turn is based on the degree to which results that they have added to staks have been subsequently recommended to, and selected, by other users; see [11] for additional information.

4 Recognising Context and Recommending Staks

In this paper we are not concerned with recommending individual result pages to HeyStaks users since this has been covered in [16] already. Rather, our focus is on the aforementioned *stak selection* task: which of a user's search staks (search case bases) are appropriate for their current search query or session. The success of HeyStaks depends critically on this. As in the example in Fig. 2, as the user searches for mountain bike related information they need to choose *Mountain Biking* as their current stak. If they do this consistently then HeyStaks will learn to associate the right pages with the right staks, and be in a position to make high quality recommendations for stak members. However, the need to manually select a stak at the start of a new search session is an extra burden on the searcher. To make this as easy as possible, HeyStaks integrates its stak-lists as part of the mainstream search engine interface (see Fig. 2) but still many users forget to do this, especially during the early stages, and this means that a majority of search sessions are associated with the searcher's default stak (*My Searches*), or an incorrect stak as part of an earlier session.

The central contribution of this paper is to provide a practical solution to this problem, one that avoids requiring the user to manually select staks at search time. To do this we draw on ideas from recommender systems, case based reasoning, and traditional information retrieval. Each stak is effectively a case base

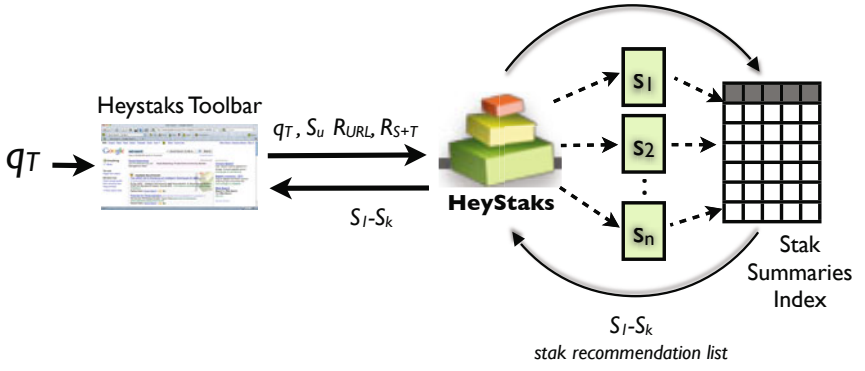

Fig. 3. Stak Recommendation

of search cases, each case representing a page that has been selected, tagged, and/or shared by stak members. For the purpose of stak recommendation we treat the combinations of the cases in each stak/case base as a type of *summary* document to reflect the stak's topic. Effectively the terms and URLs collectively contained in the stak cases become the terms of the summary document and in this way a collection of staks (case bases) can be represented as a collection of documents. Using Lucene, these documents (each one representing a single case base) can then be transformed into a *stak summary index* (or *SSI*); see Fig. 3. Then, at search time, we can use the searcher's query as a probe into this stak summary index to identify those staks/case bases that are most relevant to the query; in this work we focus only on staks that the user is currently a member of but a similar technique could be used to recommend other third-party staks in certain circumstances. These recommended staks can then be suggested directly to the user as a reminder to set their appropriate stak context; or, alternatively, we can configure HeyStaks to automatically pre-select the top ranking recommendation as the current stak context, while providing the searcher with an option to undo this if they deem the stak to be incorrect.

In the above we assume that the user's own search query (q_T) is used as the *stak query*, but in fact there are a number of additional sources of information that could be usefully harnessed for this. For example, at search time, the initial set of search engine results represents a valuable source of additional context information. Our approach to getting additional context information is similar to the technique proposed by [18,12,14], as we exploit local sources of context by using the result of a search as the basis for context assessment, extracting context terms that can then be used to augment the user's original query. However rather than use the context information in query augmentation, we are using the context to find similar staks.

Specifically we use the terms in the search engine result titles and snippets (R_{S+T}), and URLs (R_{URL}) in addition to the user's short search query. Accordingly, we refer to three basic *types* of stak recommendation strategy – *query, snippet, URL* – depending on which sources of information form the user's stak

query (S_Q). We might also consider a recommendation strategy based on the *popularity* of the stak for the user so that staks that they use more frequently are more likely to be recommended.

At stak recommendation time we use Lucene's standard TF*IDF weighting model as the basis for scoring recommended staks as shown in Equations 3 and 4. Effectively, terms in the stak summary index (SSI) representing each case base are scored based on the TF*IDF model, preferring terms that are frequent within a given case base but infrequent across the user's staks/case bases (S_U) as a whole.

$$RecList(S_Q, S_U, SSI) = \begin{array}{l} SortDesc(Score(S_Q, S, SSI)) \\ \forall S \epsilon S_U \end{array} \tag{3}$$

$$Score(S_U, S, SSI) = \sum_{t \epsilon S_U} tf(t, S) \times idf(t, SSI) \tag{4}$$

In this way we can generate different recommendation lists (RL_{URL}, RL_{query}, RL_{S+T}) by using different sources of data as the stak query (S_Q); for example, we can use the terms in result titles and snippets as the stak query, which will lead to staks being recommended because they contain lots of distinctive title and snippet terms. Of course we can also look to combine these different sources of terms, for example, by ranking recommended staks according to their position across the recommendation lists produced by different sources of query terms. For instance, we can define the *rank score* of a given stak, across a set of recommendation lists, to be the sum of the positions of the stak in the different recommendation lists with a simple penalty assigned for lists that do not contain the stak as per Equations 5 and 6. The final recommendation list is then sorted in ascending order of the rank scores of recommended staks.

$$RankScore(s, RL_1 - RL_n) = \sum_{RL_i \epsilon RL_1 - RL_n} PositionScore(s, RL_i) \tag{5}$$

$$PositionScore(s, RL) = \begin{cases} Position(s, RL) \text{ if } s \epsilon RL; \\ Length(RL) + 1 \text{ otherwise.} \end{cases} \tag{6}$$

In summary then, HeyStaks is based on the idea of search staks which are effectively case bases of search experiences. Users can be members of many staks and at search time we need to know which stak is most likely to match their current search needs, without having to ask them directly. This is a case base retrieval problem, which we address by treating the case bases themselves as cases in their own right. Each of these 'case base' cases is made up of the combination of its individual search cases. The advantage of this approach is that we can now apply a wide range of conventional retrieval techniques to help select the right case base, and therefore search stak, at search time.

This provides for a general purpose approach to stak recommendation, which accommodates different sources of query data, and provides for a flexible way to

combine multiple recommendation lists to generate an ensemble recommendation list. The intuition of course is that by combining different sources of query data we will generate better recommendations, which we shall look at in the following evaluation.

5 Evaluation

In this section we evaluate the different forms of our stak recommendation approach, based on live-user search data, and focusing in particular on the overall recommendation accuracy of the different techniques, and combinations of techniques, across different stak types.

5.1 Setup

The data for this evaluation stems from HeyStaks usage logs generated during the period October 2008 - January 2011. For the purpose of this evaluation we limit our interest to only those activities that are associated with non-default search staks submitted by 29 active users who submitted a minimum of 100 queries through HeyStaks; this means that we focused on search sessions where the user actively selected a specific stak, which we can use as the ground-truth against which to judge our techniques. The test dataset covers 4343 individual searches. The activity levels and stak memberships of these users is presented in Figure 4; we can see that the average activity level per user is 150 activities (result selections, tags, shares etc) and the average user is a member of 19.6 staks. Overall these users were members of 229 unique staks and the size of these staks (by numbers of unique URLs) is presented in Figure 5; we further divide these staks base on their relative size as shown. For the purpose of this study we evaluate different recommendation strategies based on our five basic techniques, namely,

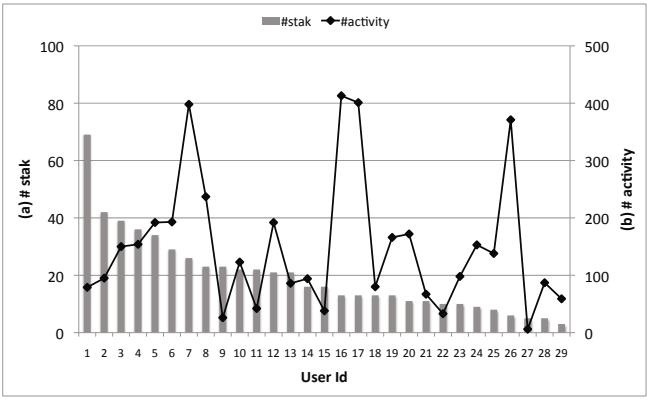

Fig. 4. Summary user data: a) a histogram of user stak membership; b) user activity levels

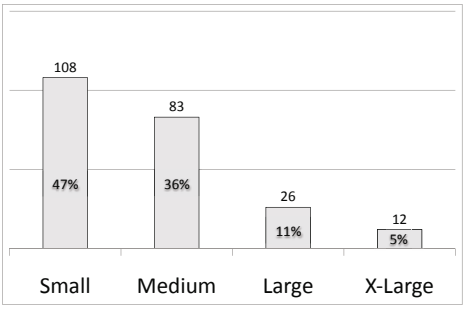

Fig. 5. A histogram of stak sizes. The staks are partitioned into groups based on their size: *small* - 1-10 pages; *medium* - 11-100 pages; *large* - 101-500 pages; *xlarge* - 500+ pages.

Query, Snippet, URL, Popularity and including all combinations of these techniques. In addition we also evaluate a baseline *random* recommendation strategy, which suggests staks at random from the user's stak-list, and also the *popularity* strategy, which recommends the user's most popular stak. This leads to a total of 16 different recommendation alternatives. To evaluate these alternatives, we generate a recommendation list for each of 4343 search instances and compute the percentage of times that the known active stak is recommended among the top k stak recommendations ($k = 1 - 5$).

5.2 Overall Recommendation Precision

To begin with we will look at the overall *success rate* across the different recommendation alternatives; in other words, how often do we recommend the correct stak (case base) to the searcher given their query? Remember we know the ground-truth from our test data since in each case the user did manually select a stak for their search. Thus by comparing the recommended staks to the ground-truth information we can calculate the success rate — how often one of the recommended staks matches the ground-truth — for the various different stak recommendation strategies and for different sizes of recommendation-lists. This data is presented in the graphs of success rate against recommendation-list size (k). For clarity we split the result data across two graphs show one set of techniques in Fig. 6 and the remaining techniques in Fig. 7.

The results indicate a wide variety of success rates across the various techniques. In Fig. 6 we can see that techniques such as *URL*, *Query*, and the combination of *URLQuery* perform poorly, recommending the correct stak about 10-20% of the time regardless of the stak-list size. In other words URL and query information does not provide a sufficiently strong signal for accurate stak recommendation on their own. In contrast, using the snippet text of results provides a much strong signal as evidenced by the superior success rates enjoyed by the *Snippet* technique in Fig. 6 , which achieves a 50% success rate when only a single stak is recommended ($k = 1$), growing to about 70% for larger recommendation-list sizes.

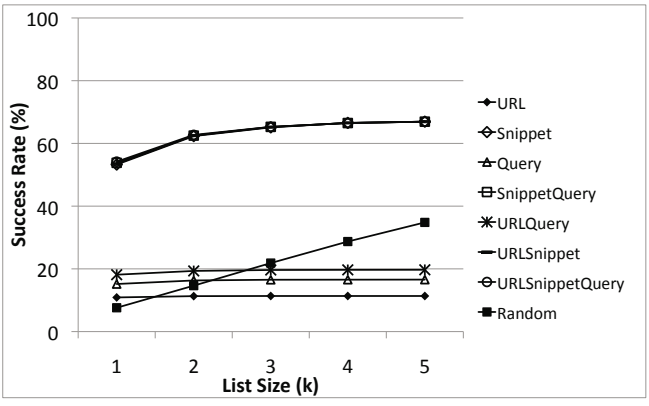

Fig. 6. Recommendation success rate for *Query, Snippet, URL* and ensemble technique

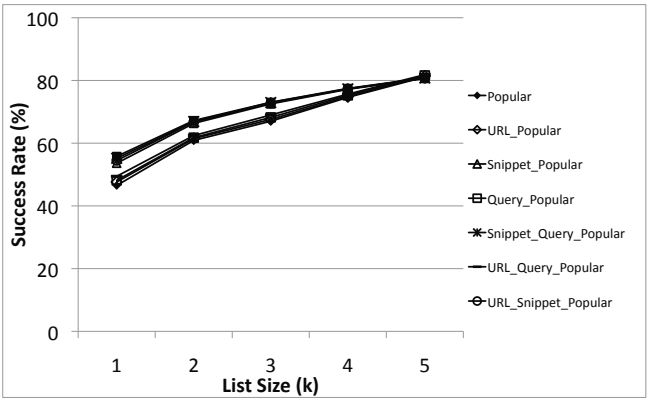

Fig. 7. Recommendation success rate for *Popularity* and ensemble technique

Fig. 6 and Fig. 7 presents the combination strategies, those that combine multiple signals during stak recommendation and we can see clear benefits across the board, with all of the combination techniques shown delivering success rates up to 80%. Importantly, we find that the combination of all signals (*Query, URL, Snippet, Popularity*) tends to deliver the best performance across different values of k. This is more clearly seen in Fig. 8 where we present the average success rate (averaged across all values of k) for the different combinations of techniques. The best performing combination combines *Query, URL, Snippet, Popularity* to achieve an average success rate of almost 71%. Interestingly, it is worth highlighting that using *Popularity* on its own delivers impressive success rates (66% on average in Fig. 8). This is in part as result of the fact that many of our test users tend to use one or two dominant staks and so a popularity-based mechanism can often make good predictions. Over time, as users spread their searches more evenly across more staks we can expect this technique to decline.

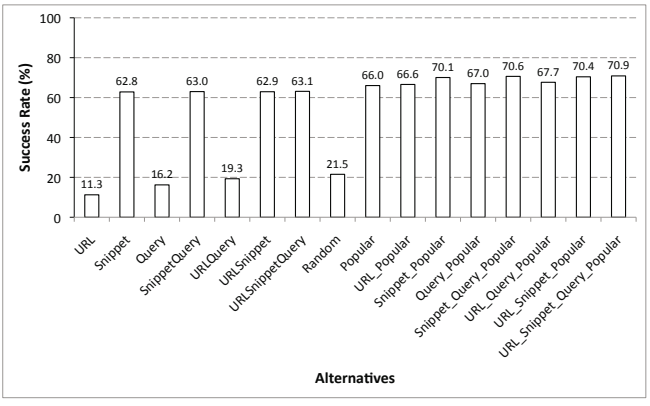

Fig. 8. Mean average success rate

Regardless, by adding additional signals to popularity we are able to further improve the stak recommendation success rate as shown.

In the above it is interesting to pay special attention to the $k = 1$ results because the ideal strategy for HeyStaks would be to automatically switch the user into a correct stak, rather than present a set of stak options for the searcher to choose from. This would require a reasonably high success rate at $k = 1$ to avoid user frustration in the case of incorrect stak switches. Unfortunately, it appears from the results in Fig. 6 and Fig. 7 that the success rates at $k = 1$ do not quite support such an automatic switching approach. For example, the best performing strategy at $k = 1$, which combines URL, *Snippet*, *Query* and *Popularity* techniques, achieves a success rate of 56%, which does not seem high enough to support an automatic stak switching.

5.3 Success vs. Stak Size

Of course the above results refer to average success rates across all staks. But not all staks are created equally. For example, as per Figure 5, the majority of staks in this study (47%) contain relatively few URLs (1-10 URLs) which surely provides a much weaker signal for their retrieval. It seems likely that this should impact on stak recommendation effectiveness when compared to larger staks. As HeyStaks matures we can expect users to develop more mature staks and so it is appropriate to evaluate the relationship between recommendation success and stak size. To test this, we present the recommendation success rate for each of the recommendation alternatives, by stak size (comparing *small, medium, large* and *extra-large* staks) for recommendation lists of size 1 (Fig. 9) and 3 (Fig. 10). In these graphs, each basic recommendation technique is represented by four separate bars showing the success rate of this technique for the four different stak types (*small, medium, large,* and *xlarge*). For convenience we also present a line-graph of the average success rate (drawn from Fig. 8) as a reference across the techniques.

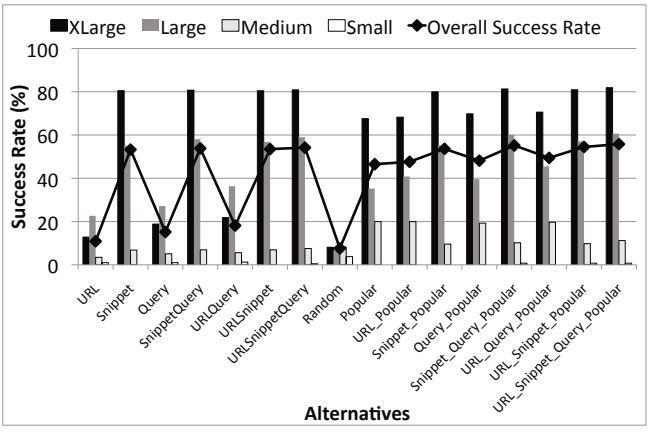

Fig. 9. Success rate by stak size where $k = 1$

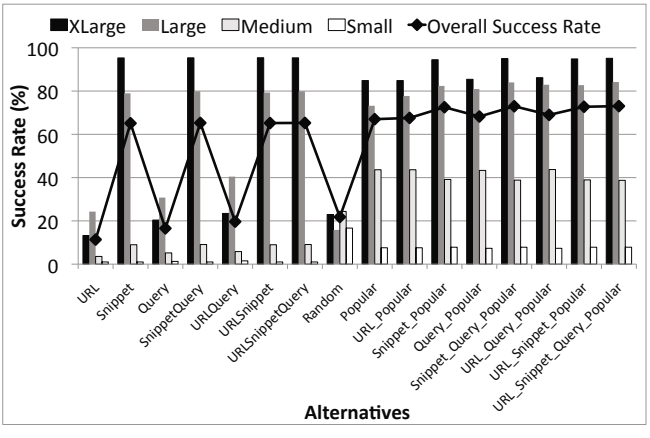

Fig. 10. Success rate by stak size where $k = 3$

It is clear that there are significant differences in recommendation accuracy across the various stak sizes. As expected the larger, more mature staks benefit from higher success rates across the various recommendation techniques and combinations. Once again the combination of all techniques does marginally better overall than any other combination. For example, looking at the combination *URL*, *snippet*, *query* and *popularity* we see a success rate of about 82% at $k = 1$ for the extra-large staks and 61% for the large staks, compared to only 11% and 1% for the medium and small staks respectively. This is encouraging because, from a engineering standpoint, it suggests that it may be practical to implement a reliable automatic stak switching policy, at least for large staks which contain more than 100 URLs. When we look at the results for $k = 3$ (see Fig. 10)

we see similar effects, only this time many combination techniques are achieving success rates in excess of 95%, for a number of recommendation combinations across the extra-large staks.

5.4 Conclusions

HeyStaks is a deployed social web search service that used ideas from case-based reasoning to help users to search more effectively online. Users can create and join so-called search staks, which are collaborative case bases of search knowledge, in order to receive community recommendations at search-time. The main contribution of this work has been to highlight a practical problem facing HeyStaks — the need to automatically predict the right stak for users at search time — and to propose and evaluate potential solutions in the form of stak recommendation strategies To this end we have described a general framework for stak recommendation, based on the indexing of staks. The approach accommodates a variety of different recommendation alternatives, using different types of query data at search time, such as search query terms, the titles, URLs, and snippet terms of search results, for example. We have described the results of a comprehensive evaluation of a variety of recommendation strategies, based on live user search data. The success rates achieved for the larger staks in particular speak to the potential for a reliable automatic stak switching mechanism, and at the very least it is possible to generate a short-list of stak recommendations that are accurate up to nearly 95% of the time.

One important limitation of this work is that recommendations are based on a single query (or the results of a single query), when we know that most searchers engage in sessions that extend beyond a single query; we will often submit 2 or 3 queries variations before we find what we are looking for. Obviously, by looking at an extended search session it may be possible to leverage more data (more queries, more URLs, more snippets) in order to better guide stak recommendation. So, for example, while it might not be possible to recommend the correct stak on the first query, we may find that the addition of a second query (and its associated URLs and snippets) greatly improves recommendation quality. Another opportunity could see the use of third-party services to enhance our understanding of the user's search context. For example, by leveraging WordNet or Wikipedia it may be possible to elaborate a users query and to better identify context and this context information could also be used for stak recommendation. These ideas will form the basis for the next steps in this research.

Acknowledgement. This work is supported by Science Foundation Ireland under grant 07/CE/I1147, HeyStaks Technologies Ltd, Ministry of Higher Education Malaysia and Universiti Teknikal Malaysia Melaka.

References

1. Balfe, E., Smyth, B.: Case-based collaborative web search. In: Funk, P., González Calero, P.A. (eds.) ECCBR 2004. LNCS (LNAI), vol. 3155, pp. 489–503. Springer, Heidelberg (2004)

2. Boydell, O., Smyth, B.: Enhancing case-based, collaborative web search. In: Weber, R.O., Richter, M.M. (eds.) ICCBR 2007. LNCS (LNAI), vol. 4626, pp. 329–343. Springer, Heidelberg (2007)
3. Briggs, P., Smyth, B.: Provenance, trust, and sharing in peer-to-peer case-based web search. In: Althoff, K.-D., Bergmann, R., Minor, M., Hanft, A. (eds.) ECCBR 2008. LNCS (LNAI), vol. 5239, pp. 89–103. Springer, Heidelberg (2008)
4. Burke, R., Hammond, K., Kulyukin, V., Tomuro, S.: Question Answering from Frequently Asked Question Files. AI Magazine 18(2), 57–66 (1997)
5. Godoy, D., Amandi, A.: PersonalSearcher: An Intelligent Agent for Searching Web Pages. In: Monard, M.C., Sichman, J.S. (eds.) SBIA 2000 and IBERAMIA 2000. LNCS (LNAI), vol. 1952, pp. 43–52. Springer, Heidelberg (2000)
6. Hatcher, E., Gospodnetic, O.: Lucene in action. Manning Publications (2004)
7. Kanawati, R., Jaczynski, M., Trousse, B., Andreloi, J.-M.: Applying the Broadway Recommendation Computation Approach for Implementing a Query Refinement Service in the CBKB Meta-search Engine. In: Conférence Française sur le Raisonnement á Partir de Cas, RáPC'99 (1999)
8. Leake, D.B., Sooriamurthi, R.: Automatically selecting strategies for multi-case-base reasoning. In: Craw, S., Preece, A.D. (eds.) ECCBR 2002. LNCS (LNAI), vol. 2416, pp. 204–233. Springer, Heidelberg (2002)
9. Leake, D.B., Sooriamurthi, R.: Managing multiple case bases: Dimensions and issues. In: FLAIRS Conference, pp. 106–110 (2002)
10. Lenz, M., Ashley, K.: AAAI Workshop on Textual Case-Based Reasoning, AAAI Technical Report WS-98-12 (1999)
11. McNally, K., O'Mahony, M.P., Smyth, B., Coyle, M., Briggs, P.: Towards a reputation-based model of social web search. In: IUI 2010: Proceeding of the 14th International Conference on Intelligent User Interfaces, pp. 179–188. ACM, New York (2010)
12. Mitra, M., Singhal, A., Buckley, C.: Improving automatic query expansion. In: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1998, pp. 206–214. ACM, New York (1998)
13. Rissland, E.L., Daniels, J.J.: A hybrid CBR-IR Approach to Legal Information Retrieval. In: Proceedings of the 5th International Conference on Artificial Intelligence and Law, pp. 52–61. ACM Press, New York (1995)
14. Shen, D., Pan, R., Sun, J.-T., Pan, J.J., Wu, K., Yin, J., Yang, Q.: Query enrichment for web-query classification. ACM Trans. Inf. Syst. 24, 320–352 (2006)
15. Smyth, B., Briggs, P., Coyle, M., OMahony, M.: A case-based perspective on social web search. In: McGinty, L., Wilson, D. (eds.) ICCBR 2009. LNCS, vol. 5650, pp. 494–508. Springer, Heidelberg (2009)
16. Smyth, B., Briggs, P., Coyle, M., O'Mahony, M.P.: Google shared. a case-study in social search. In: User Modeling, Adaptation and Personalization, pp. 283–294 (2009)
17. Sooriamurthi, R.: Multi-case-base reasoning. PhD thesis, Indiana University, Indianapolis, IN, USA, AAI3278223 (2007)
18. Xu, J., Croft, W.B.: Query expansion using local and global document analysis. In: Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1996, pp. 4–11. ACM, New York (1996)