Abstract. In this paper we focus on a multi-case case-based reasoning system to support users during collaborative search tasks. In particular we describe how repositories of search experiences/knowledge can be recommended to users at search time. These recommendations are evaluated using real-world search data.

Keywords: social search, context recommendation

1 Introduction

HeyStaks is a social search service that is designed to work with mainstream search engines via a browser plugin. The main techniques have been described elsewhere [13, 14] but very briefly HeyStaks makes result recommendations to searchers at search time, based on the past searches of their social network; see Figure 1. 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 [13] 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. A key problem for HeyStaks is to ensure that the right stak is chosen for a given search session and
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.

in this paper we describe how information about the search sessions of users can be used to automatically recommend staks at search time.

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; see for example [11, 10, 7, 4]. 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 [1–3] 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 present work is the fundamental role that multiple case bases play in search support; each search stak is an individual search case base. As such it is related to the work of, for example, [15] which introduced the idea of multi-case-base reasoning (MCBR) and proposed a novel distributed case-based reasoning architecture to supplement local case base knowledge; see also [9, 8]. In the present paper we are also concerned with a form of multi-case
base reasoning. As above, our case bases 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.

2 Recognising Context & Recommending Staks

In this paper we are not concerned with recommending individual result pages to HeyStaks users since this has been covered in [14] 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.

2.1 Representing the Index Stak

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 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. 2.
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.

2.2 Recommending Staks

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. 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.

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 1 and 2. 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.

\[
RecList(S_Q, S_U, SSI) = \text{SortDesc}(\text{Score}(S_Q, S, SSI)) \quad \forall S \in S_U
\]

\[
\text{Score}(S_U, S, SSI) = \sum_{t \in S_U} tf(t, S) \times \text{idf}(t, SSI)
\]

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 3 and 4. The final recommendation list is then sorted in ascending order of the rank scores of recommended staks.

\[ \text{RankScore}(s, RL_1 - RL_n) = \sum_{RL_i \in RL_1 - RL_n} \text{PositionScore}(s, RL_i) \quad (3) \]

\[ \text{PositionScore}(s, RL) = \begin{cases} \text{Position}(s, RL) & \text{if } s \in RL; \\ \text{Length}(RL) + 1 & \text{otherwise.} \end{cases} (4) \]

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 a general purpose approach to stak recommendation, which accommodates different sources of query data, and provides 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 is already shown in [12].

### 2.3 Harnessing Extended Sessions

Our initial stak recommendation work presented in [12] is limited in one important respect. Recommendations are based on singleton search sessions, that is, a single query submission. This limit the amount of data that is available as the the stak query \((S_Q)\) which drives the recommendation process because we only have access to a single query and a single result-list. It is well known, however, that most search sessions are made up of multiple queries [16] as searchers modify and refine their queries in order to find the precise results they need.

Thus, in this work we will investigate the use of these extended search sessions during stak recommendation. For example, while it might not be possible to reliably recommend the correct stak on the first query, the addition of a second query (and its associated URLs and snippets) may improve recommendation quality. In order to group multiple queries into one session, first we need to identify the session boundary. For the purpose of defining session boundary we use the method introduced by Jansen et. al [6]. In this method, we use user id and common terms across the queries for session identification and group the common queries into a session (see Algorithm 1).
Algorithm 1 Algorithm for sessions identification

**Assumptions:** Records of usage log is sorted by User Id and time (ascending order)

**Input:** Record \((R_i)\) with query \((Q_i)\), User Id \((U_i)\), Session Id \((S_i)\), Record \((R_{i+1})\) with query \((Q_{i+1})\), User Id \((U_{i+1})\)

**Output:** Session Id \((S_{i+1})\) for \(R_{i+1}\)

Begin
  if \(U_i \neq U_{i+1}\) then
    \(S_{i+1} = S_i + 1\)
  else
    \(B = \{t|t \in Q_i \land t \in Q_{i+1}\}\) // no. of terms in common
    if \(B > 0\) then
      \(S_{i+1} = S_i\)
    else
      \(S_{i+1} = S_i + 1\)
  end if
end if
End

As a result our stak query \((S_Q)\), as defined in the previous section, can now be adapted to cover sessions with multiple queries. And in turn we can apply the same stak recommendation techniques on these extended stak queries to investigate their impact on overall recommendation accuracy.

3 Evaluation

In this section we evaluate our extended stak recommendation technique on search data collected from the HeyStaks system. For the purpose of this paper we will focus mainly on the relationship between overall recommendation accuracy and session length (that is, the number of queries per session).

3.1 Data Set

Our dataset comes from the HeyStaks usage logs generated during the period October 2008 - January 2011 across a group of the 28 active users, who are members of approximately 20 staks each, and who have each submitted at least 100 queries. For the purpose of this evaluation we limit our interest to only those sessions that are associated with at least one non-default search staks so that we can focus on search sessions where the user actively selected a specific stak. This selected stak is used as the ground-truth against which to judge our recommendation techniques. The test dataset covers 10,177 individual searches which have been grouped into 4,545 sessions (according to Algorithm 1). This provides an average session length of 2.2 queries and an average session count per user of 162, so in average each session consist of 2.2 queries.
3.2 Session Data

Each session consists of one or more queries and it is interesting to look at the relationship between consecutive queries, \( q_i \) and \( q_{i+1} \), from the same user and session to get a sense of the type of modifications and refinements that searchers tend to make. The following modification classes are based on those presented by [5]:

- **New** – initial query in a session.
- **Reformulation** – current query is on the same topic as the previous query and both queries contain common terms. (add some terms and removed others from the previous query and both queries still have some common terms)
- **Generalization** – current query is in the same topic as previous query, but the searcher seeking more general information (remove terms from previous query)
- **Specialization** – current query is on the same topic as the previous query, but the search is now seeking more specific information. (adding new terms to the query)
- **Change Stak** – current query same as previous query, but the stak has been changed

Fig. 3(a) presents the frequency of these different modification types. We can see that users frequently change their staks during a session (19% of the time). We also see that overall about 36% of the modifications involve changes to the terms in the query, which will ultimately lead to changes in the result-list return to users, and so provides a strong indication that leveraging these extended sessions will deliver a richer source of information for stak recommendation. In Fig. 3(b) we present the unique number of query and snippet terms, and URLs, for different session lengths and we can see that there is a steady increase in the quantity of this information as session length grows. However, it is worth highlighting that, for example, doubling the session length, does not deliver twice the number of unique query or snippet terms or unique URLs; the reason being, of course, that minor modifications to the query will not radically change the new result-list.

3.3 Precision vs. Session Length

In this paper we are primarily concerned with the accuracy of our stak recommendation techniques. As described earlier we have three basic recommendation strategies, namely **Query**, **Snippet**, **URL**, so that our stak query is composed of either query, snippet or URL information. And, as in [12] we will also consider combinations of these techniques. In addition we also evaluate a baseline stak **popularity** strategy, which recommends the user’s most popular stak. All of this leads to a total of 15 different recommendation alternatives, but for the purpose of this evaluation, and for reasons of space, we will focus on 7 of these alternatives.
To evaluate these alternatives, we generate a recommendation list for each of 4,545 search session instances and compute the percentage of times (success rate) that the known active stack (ground-truth) is recommended among the top $k$ stack recommendations (here we look at $k = 1$ and $k = 3$). We calculate this success rate across sessions of different lengths and the results are presented in Fig. 4 and Fig. 5 for $k = 1$ and $k = 3$ conditions.

The results indicate a wide variety of success rates across the various techniques and session lengths. In both Fig. 4 and Fig. 5 we can see that techniques such as URL, Query, and the combination of URLQuery perform poorly, recommending the correct stack about 32-47% ($k = 1$) and 37-51% ($k = 3$) of the time across different session length. In other words URLs and query information does not provide a sufficiently strong signal for accurate stack recommendation on their own. In contrast, using Snippet technique on its own provides a much stronger signal with 54-64% ($k = 1$) and 83-88% ($k = 3$). The combination of all signal (URL, Query, Snippet, Popularity) provides the best performance in most of the time when $k = 1$ which achieves about of 55-67% success rate. However, when $k = 3$ the best technique is the combination of URLs, queries and snippets (URLSnippetQuery) terms with success rate between 87% and 91%.

In the above it is interesting to pay special attention to how success rate changes with session length, after all the core hypothesis in this work is that by harnessing longer sessions we will improve stack recommendation accuracy. According to the results presented this hypothesis is supported but only to a certain extent. For example, the best performing techniques show some improvement as we move from sessions of length 1 to sessions of length 2 but beyond this session length delivers only marginal success rate benefits. For example, for URLSnippetQueryPopularity technique at $k = 1$ we can see that success rate grows from 55% for singleton sessions to 66% for sessions of length 2, but subsequent session information only serves to deliver minor success rate gains up to about 67%. So although, longer sessions do deliver additional data to drive recommendation the extent to which this can be leveraged, at least in this stack recommendation approach is limited. Nonetheless, the success rate improvements for sessions of length 2 are certainly worthwhile and the fact that they occur for shorter ses-

Fig. 3. Summary of session data; (a) query modification classes (b) average unique URLs, query terms and snippet terms across different session length
Fig. 4. Recommendation success rate for each session length where $k = 1$

Fig. 5. Recommendation success rate for each session length where $k = 3$

sessions means that these improvements will be accessible across a much greater variety of search sessions, since most sessions are shorter in length.

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 build on recent potential solutions in the form of stak recommendation strategies. We have described and evaluated a recommendation technique that leverages information available across extended search sessions and demonstrated significant improvements in recommendation accuracy.
Acknowledgement. This work is supported by Science Foundation Ireland under grant 07/CE/11147, HeyStaks Technologies Ltd, Ministry of Higher Education Malaysia and Universiti Teknikal Malaysia Melaka.

References