Web Search Intent Induction via Search Results Partitioning

Hal Daumé III
University of Southern California
Department of Computer Science
Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292
hdaume@isi.edu

Eric Brill
Microsoft Research
One Microsoft Way
Seattle, WA 98052
brill@microsoft.com

Abstract

We present a computationally efficient method for automatic clustering of web search results based on partitioning the return set according to queries the user may have intended. The method requires no data other than query logs and the standard inverted indices used by most search engines. Our method outperforms standard web search in the task of enabling users to quickly find relevant documents for informational queries.

1 Introduction and Motivation

In a study of web search query logs, Broder (2002) observed that most queries fall into one of three basic categories: *navigational*, *informational* and *transactional*. A navigational query is one where the user has a particular URL they are attempting to find. An informational query is one where the user has a particular information need to satisfy. A transactional query is one in which the user seeks to perform some sort of web-mediated activity (such as purchasing a product). Arguably, a fourth category exists, *browsing queries*, where a user is simply using a search engine to direct their web browsing, though Broder does not distinguish these types of searches.

In that paper, Broder (2002) also confirms that most queries are very short: on the order of two words. For navigational queries, this does not prevent search engines from performing well: experience tells us that attempting to find Eugene Charniak's web page by searching for "Eugene Charniak" or even just "Charniak" will be successful. This is likely largely due to the fact that when a user has a particular URL they wish to find, they know a lot about it and can easily invent queries which are sufficiently discriminative. Unfortunately, according to Broder, only 20 to 24.5 percent of all web queries are navigational.

On the other hand, for informational (and browsing) queries, it is often the case that the user does not know exactly what sort of information he is looking for. This makes it difficult for him to formulate enough, or even correct, keywords to represent his need. These types of queries make up anywhere between 39 and 48 percent of all web queries, according to Broder, making them a prime target for research.

2 Prior Work

Our interest is in informational and browsing queries. The general approach we are exploring to assist users in finding what they want for these sorts of queries is to present structured results. Dumais et. al. (2001) have shown that presenting results in a structured fashion improves a user's ability to find relevant documents quickly.

There are three general techniques for presenting web search results in a structured manner, ranging from totally supervised methods to totally unsupervised methods.

The first approach, *manual classification*, is typified by a system like Yahoo!. In these systems, humans have created a hierarchical structure describing the web and manually classify web pages into this hierarchy. One advantage of manual classification systems is accuracy. Another is that they have meaningful structure and thus the labellings on different nodes in the hierarchy are meaningful. Unfortunately, they are also expensive and time consuming to build, they never achieve full coverage, and they cannot adapt to different user needs.

The second approach, *automatic classification* (see, for instance, the classification system reported by Dumais (2000)) builds on the hierarchies constructed for manual classification systems, but web pages are categorized by a (machine-learned) text classification system. These achieve full coverage, but are less accurate than manual classification systems. They are less expensive (humans need not constantly insert new web pages into the hierarchy), but still costly as the original hierarchy needs to be built. They are necessarily static and cannot readily morph to fit the needs of different users; neither can they adapt with time, without considerable expense.

The third approach, typified by systems such as Vivisimo and Lighthouse (Leuski and Allan, 2000), as well as the system of Zamir et al. (1999), look at the text of the returned documents and perform document *clustering*. These approaches trade off economic cost for computational cost: there is no hierarchy to be built by hand, but clustering based on document text is a computationally expensive operation. Moreover, once documents are clustered, appropriate names must be assigned to these clusters, which is an on-going research challenge.

Another approach to this problem is from Beeferman and Berger (2000). Their system leverages click-through data to cluster related queries. The intuition behind their method is that if two different queries lead to users clicking on the same URL, then these queries are related. They use the click-through data to define a bipartite graph and then use agglomerative clustering to group similar queries.

Our approach is most closely related to the approach of Beeferman and Berger (2000), but does not require click-through data. Moreover, the use of click-through data can result in query clusters with low user utility.[1] Furthermore, our approach does not suffer from the computation cost of document clustering by text and produces structured results with meaningful names like Yahoo! without the economic cost of building hierarchies by hand.

3 Problem Specification

Underspecified queries are a prevalent and challenging problem for web search. When a user enters an underspecified informational or browsing query, we wish to predict the true information need and present these results to the user. Unfortunately, by definition, there are many possible information needs for a single underspecified query. Therefore, we attempt to provide a range of possible needs to the user.

Our general strategy is the following: when a user, John, issues an underspecified query to a standard search engine, he will likely be presented with a long list of very similar, only marginally relevant documents. Perhaps, at some time in the past, a more knowledgeable user, Mary, has had the same search need. However, since Mary knew more about this shared search need than John, Mary was better able to formulate a query. This, we may reasonably assume, resulted in her search being more successful. Our goal is to use Mary's expertise to help John.

For instance, John may be a beginning fly fishing enthusiast. Perhaps he is going fly fishing for trout soon and needs to find information about what sort of flies to purchase. To this end, he queries his favorite search engine for "fly fishing." Due to the fact that this is a grossly underspecified query, John will likely have to read through pages of results before actually finding information on what he really wants.

[1]See Section 4.3

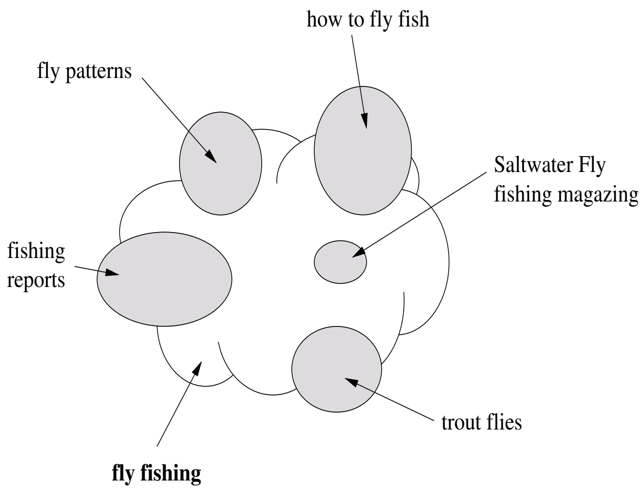

Figure 1: Related queries to "fly fishing".

However, it might be the case that Mary is also a fly fishing enthusiast and knows that you are supposed to use a different sort of fly to catch trout. She needed information on these flies and so she searched for "trout flies." Since this query is more specific, she was able to obtain good results. If we were able to realize that these two queries are related, we might be able to help John perform a more informed search.

4 Methodology

Suppose John enters his query for "fly fishing." This will retrieve a large set of documents, depicted by the cloud in Figure 1. We assume that John's search need (information about flies for catching trout) is somewhere in or near that cloud, but we do not know exactly where. However, we can attempt to divide up this space in a meaningful way. In particular, we can look for other queries, made by other people, which return an overlapping set of documents. The return sets for these other queries are represented in Figure 1 by shaded circles. We refer to this process as *Query Driven Search Expansion* and henceforth refer to our system as the QDSE system.

4.1 Formal Specification

Formally, if Q is the set of queries to our search engine and D is the set of indexed documents, let R be a binary relation on $Q \times D$ where qRd if and only if d is in the return set for the query q. Then, given a query q, we are looking for all queries q' such that $(\exists d \in D)(qRd \wedge$

$q'Rd$). More briefly, we are interested in the set $(R^{-1} \circ R)[q]$.

It is likely that the set of $(R^{-1} \circ R)[q]$ is quite large for a given q (in practice the size is on the order of ten thousand; for our dataset, "fly fishing" has $29,698$ related queries). It is infeasible to present this entire set to the user. However, some of these queries will be only tangentially related to q. Moreover, some of them will be very similar to each other. In order to measure these similarities, we define a distance metric[2] between two queries q and q' based on their returned document sets:

$$\|q, q'\| = 1 - \frac{|R[q] \cap R[q']|}{|R[q] \cup R[q']|} \quad (1)$$

One could then sort the set of related queries according to $\|q, q'\|$ and present the top few to the user. Unfortunately, this is insufficient: the top few are often too similar to each other to provide any new useful information. To get around this problem, we use the maximal marginal relevance (MMR) scheme originally introduced by Carbonell et. al. (1998).

MMR was originally introduced to rerank documents in a pure information retrieval setting. Its goal was to return documents which are simultaneous *relevant* and *original.* In their setup, they defined a similarity metric between documents and queries (s_q), as well as a similarity metric between documents and other documents (s_d). They then iteratively selected documents according to the following weighting:

$$\operatorname*{argmax}_{d \in D} \left[\lambda s_q(q, d) - (1 - \lambda) \max_{d' \in D'} s_d(d, d') \right] \quad (2)$$

In this equation, D is the set of documents not yet returned and D' is the set of documents already returned. λ is an interpolation factor to be set ahead of time. This equation then selects the document d which is maximally similar to the query q and simultaneously maximally different from other documents already returned, d'. For instance, when $\lambda = 1$, the ranking is simply according to the similarity between the document and the query. When $\lambda = 0$, the algorithm attempts to produce maximally different results, independent of what the query was.

Our situation is very similar. We wish to present the user with queries q' which are related to their original query q, but different from other queries already returned. Converting Equation 2 to use with distances instead of similarities and to use queries instead of documents, we order according to:

$$\operatorname*{argmin}_{q'} \left[\lambda \|q, q'\| - (1 - \lambda) \min_{q''} \|q', q''\| \right] \quad (3)$$

[2]It is easy to verify that this satisfies the three required properties of a distance metric, but we do not do this here.

where the q's are drawn from the unreturned query expansions and the q''s are drawn from the previously returned set.

4.2 Data Cleanup

As is typical in applications dealing with natural language, there are a few preprocessing steps which need to be applied to the set of related queries in order for them to be presentable for human consumption.

First, queries q' which are lexically too similar to the original query q are thrown out. This step removes queries whose words are simply a permutation of the words of the original query (after removing stop words). Since these will have nearly identical return sets, they will be chosen first according to Equation 3, but will provide no new information.

Secondly, queries which look like URLs (that contain any of the strings "http", "www", ".com", ".net", ".edu", or ".org") are thrown out. These queries are usually mistakes where people meant to type the URL into the address bar, but focus was stolen unknown to them and placed in the query box. The return sets are typically unmeaningful and suggesting a URL as an alternative query to a user can be confusing.

Thirdly, we throw out queries which are not at least as long (in number of words) as the original query. This is not strictly necessary, but allows us to give some sort of refinement process.

Finally, we throw out all queries which contain characters which are not printable ASCII. Since a non-trivial fraction of the queries in our log are in non-Latin scripts (predominant are Chinese and Japanese), it is worthwhile removing these when evaluating with an English-speaking audience.

4.3 Alternative Distance Metrics

One particular thing to note in Equation 1 is that we do not take relative rankings into account in calculating distance. That is, if a document d was the first result returned for two queries q and q', this does not allow it to influence the distance measurement any more than if it were the 100th result.

One could define a distance metric weighted by each document's position in the return list. The typical way to define such a metric is:

$$\|q, q'\|_2 = 1 - \frac{\sum_{d \in R[q] \cap R[q']} w(d)}{\sum_{d \in R[q] \cup R[q']} w(d)} \quad (4)$$

where w is a query-independent weighting function for a document (for instance, this might be the document's PageRank (Brin and Page, 1998)). Equation 1 is a special case of this where $w(d) = 1$ for all documents d. So long as the range of w is strictly positive, the function defined in Equation 4 is a distance metric.

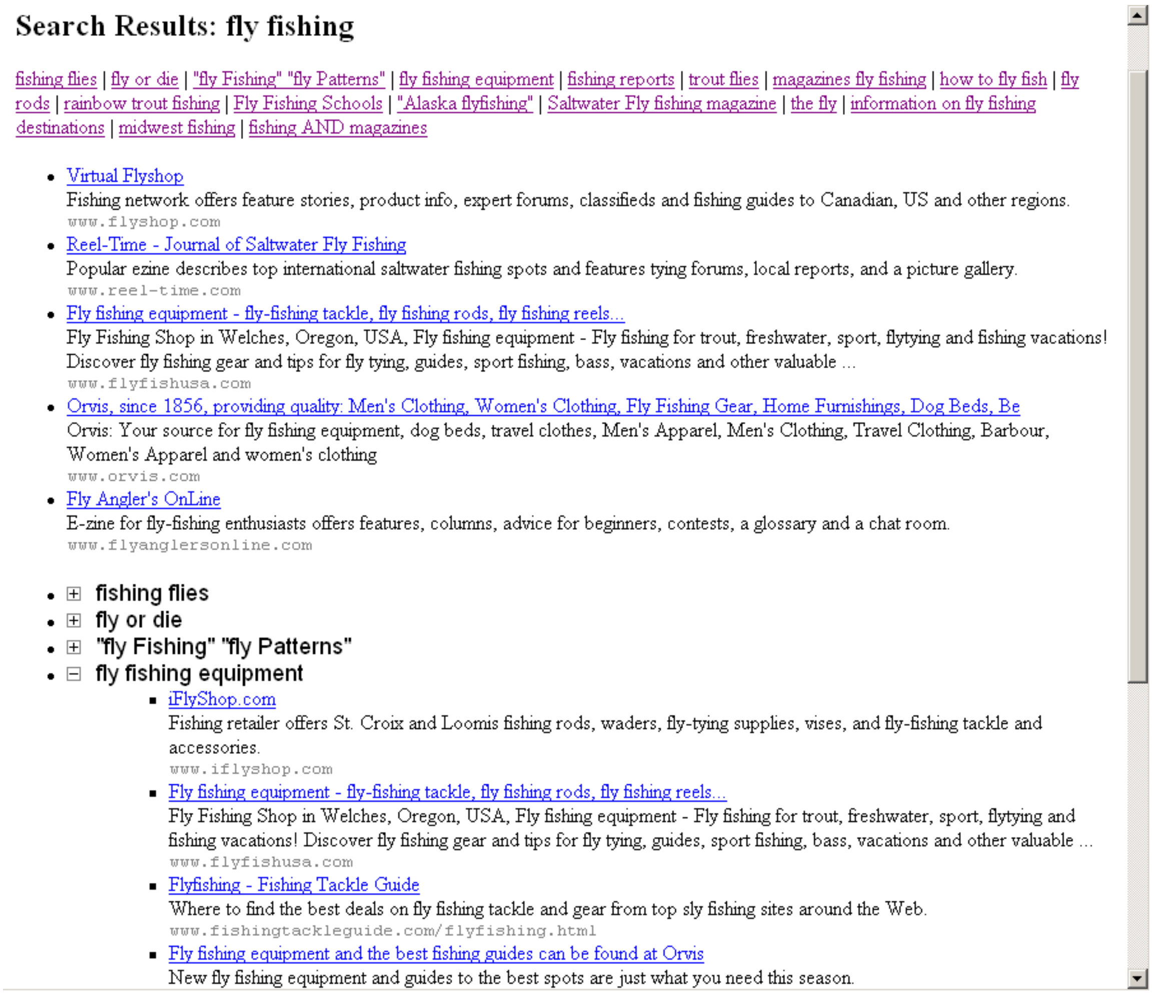

Figure 2: Screenshot of system output for the query "fly fishing".

We ran experiments using this distance metric where w was PageRank. The results of this distance metric were inferior to the standard (Equation 1) ranking. We attribute this degradation to the fact that if two queries agree only on their top documents, they are too similar to be worth presenting to the user as alternatives. This is directly related to a weakness of the Beeferman and Berger (2000) approach: by using click-through data, their approach is nearly equivalent to using the PageRank metric in Equation 4 (if one assumes PageRank is a reasonable approximation for relevance). Thus, their system essentially only learns synonymy between queries, rather than relatedness. While synonymy is interesting for the search back-end to understand, it makes little sense to present to the user a list of synonymous queries to select from, because synonymous queries will likely return too similar results. We would much rather present them with a list of specifications.

5 System

The system described above functions in a completely automatic fashion and responds in real-time to users queries. A screen shot of the interface for the query "fly fishing" – errors and all – is shown in Figure 2.

Across the top of the return results, the query is listed, as are the top ranked q' in $(R^{-1} \circ R)[q]$. Each of these query suggestions is a link to a heading, which are shown below. Below this list are the top five search result links from MSN Search under the original query. These are included because there is a chance the user knew what he was doing and actually entered a good query.

After the top five results from MSN Search, we display each header with a +/- toggle to expand or collapse it. In Figure 2, the suggested query "fly fishing equipment" is expanded. Under each expanded query we list the top 4 results from that query.

6 Evaluation Setup

Evaluating the results of search engine algorithms without embedding these algorithms in an on-line system is a challenge. We evaluate our system against a standard web search algorithm (in our case, MSN Search). Ideally, since our system is focused on informational queries, we would like a corpus of $\langle \text{query}, \text{intent} \rangle$ pairs, where the query is underspecified.

One approach would be to create this corpus ourselves. However, doing so would bias the results. An alternative would be to use query logs; unfortunately, these do not include intents. In the next section, we explain how we create such pairs.

6.1 Deriving Query/Intent Pairs

The solution we arrived at is the following: we have a small collection of click-through data, based on experiments run at Microsoft Research over the past year. This data was collected by tracking search usage of volunteer Microsoft employees over several months. Each query made by one of these volunteers to MSN search was recorded together with a list of the URLs of all the result links followed and the amount of time spent on that page.

Given this data, for a particular user and query, we look for the last URL they clicked on and viewed for at least two minutes, so long as this URL is not in the top five results. We consider all of these documents to be satisfactory solutions for the user's search need. We discard results that were in the top five because we intend to use these pairs to evaluate our system against vanilla MSN Search. Since the first five results our system returns are identical to the first five results MSN Search returns, it is not worthwhile annotating these data-points.

These $\langle \text{query}, \text{URL} \rangle$ pairs give us a hint at how to get to the desired $\langle \text{query}, \text{intent} \rangle$ pairs. For each $\langle \text{query}, \text{URL} \rangle$ pair, we looked at the query itself and the web page at the URL. We also knew that none of the top five MSN Search results satisfied the user, so we also look at those.[3]

Given the query, the relevant URL and the top five MSN Search results, we attempted to create a reasonable search intent that was (a) consistent with the query and the URL, but (b) not satisfied by any of the top five results. There were a handful of cases where we could not think of a reasonable intent for which (b) held – in these cases, we discarded that pair.

In all, we created 52 such pairs; ten arbitrarily chosen $\langle \text{query}, \text{URL}, \text{intent} \rangle$ triples are shown in Table 1.

[3] It may be the case that the users found an earlier URL also to be relevant. This does not concern us, as we do not actually use these URLs for evaluation purposes – we simply use them to gain insight into intents.

6.2 Relevance Annotation

Our evaluation now consists of giving human annotators $\langle \text{query}, \text{intent} \rangle$ pairs and having them mark the first relevant URL in the return set (if there is one). However, in order to draw an unbiased comparison between our system and vanilla MSN Search, we need to present the output from both as a simple ordered list. This requires first converting our system's output to a list.

6.2.1 Linearization of QDSE Output

We wish to linearize our results in such a way that the position of the first relevant URL enables us to draw meaningful inferences. In vanilla MSN search, we can ascribe a cost of 1 to reading each URL in the list; then, having a relevant URL as the 8th position results in a cost of 8.

Similarly, we wish to ascribe a cost to each item in our results. We do this by making the assumption that the user is able to guess (with 100% accuracy) which subcategory a relevant URL will be in (we will evaluate this assumption later). Given this assumption, we say that the cost of a link in the top 5 vanilla MSN links is simply its position on the page. Further down, we assume there is a cost for reading each of the MSN links, as well as a cost for reading each header until you get to the one you want. Finally, there is a cost for reading down the list of links under that header.

For example, referring back to Figure 2, if the relevant link were "Fly Angler's OnLine", the cost would be 5 (because you would have had to have read five lines to get that far). On the other hand, if the relevant link were "Flyfishing - Fishing Tackle Guide", the cost would have been $5 + 4 + 3 = 12$ (the five from reading the vanilla MSN links, the four for reading the first four headers, and the three for reading three links in that list).

Given this cost model, we can linearize our results by simply sorting them by cost (in this model, several links will have the same cost – in this case, we fall back to the original ordering). We can then compare the average cost for finding documents in our system against the average cost for finding documents in the vanilla MSN system (again, under the assumption that a user will be able to guess the appropriate category).

6.2.2 Annotation

We divided the 52 $\langle \text{query}, \text{intent} \rangle$ pairs into two sets of 32 (there were 12 pairs common to both sets). Each set of 32 was then scrambled and half were assigned to class *System 1* and half were assigned to class *System 2*. It was ensured that the 12 overlapping pairs were half assigned to *System 1* and half assigned to *System 2* in both cases.

Four annotators were selected. The first two were presented with the first 32 pairs and the second two were presented with the second 32 pairs. The first and second

Query: Soldering iron	**URL:**	`www.siliconsolar.com/accessories.htm`	
Intent:	looking for accessories for soldering irons (but not soldering irons themselves)		
Query: Whole Foods	**URL:**	`www.wholefoodsmarket.com/company/communitygiving.html`	
Intent:	looking for the Whole Foods Market's community giving policy		
Query: final fantasy	**URL:**	`www.playonline.com/ff11/home/`	
Intent:	looking for a webforum for final fantasy games		
Query: online computer course	**URL:**	`www.microsoft.com/traincert/`	
Intent:	looking for information on Microsoft Certified Technical Education centers		
Query: sailing pictures	**URL:**	`www.waterways.com/wboats.html`	
Intent:	looking for a site which has pictures of many makes and models of sailboats		
Query: Plesiosaurs	**URL:**	`www.ucmp.berkeley.edu/history/plesio.html`	
Intent:	looking for a site with hoaxes about the connection between the Loch Ness monster and Plesiosaurs		
Query: baseball diamond	**URL:**	`www.oski.org/html/dir_baseball.htm`	
Intent:	looking for directions to Evans Baseball Diamond		
Query: smart tags	**URL:**	`msdn.microsoft.com/library/en-us/dnofftalk/html/office01022003.asp`	
Intent:	looking for a smart tags document describing their use in Office XP		
Query: stone veneer	**URL:**	`www.products-furniture-decor.com/stone_products/stone_siding.html`	
Intent:	looking for literature on stone veneer products		
Query: ATI graphics	**URL:**	`www.atitech.ca/products/pc/ragefury/`	
Intent:	looking for support information for the Rage Fury graphics card by ATI		

Table 1: 10 random $\langle$query, URL, intent$\rangle$ triples

annotator, therefore, were annotating exactly the same $\langle$query, intent$\rangle$s, but the systems were swapped in each case. So for annotator 1, *System 1* was vanilla MSN and *System 2* was QDSE; while for annotator 2, *System 1* was our system and *System 2* was vanilla MSN.[4] The same was done with the second pair of annotators. Annotators were given a query, the intent, and the top 100 documents returned from the search according to the corresponding system. They selected the first link which answered the intent. If there were unable to find a relevant link, they recorded that information as well.

The result of this annotation is that we have 52 $\langle$query, intent$\rangle$ pairs with the associated cost of the first relevant document for each system. Furthermore, we have 12 overlapping pairs on which we can calculate inter-annotator agreement.

6.3 Predictivity Annotation

Our cost function for the linearization of the hierarchical results (see Section 6.2.1) assumes that users are able to predict which category will contain a relevant link. In order to evaluate this assumption, we took our 52 queries and the automatically generated category names for each using the QDSE system. We then presented four new annotators with the queries, intents and categories. They selected the first category which they thought would contain a relevant link. They also were able to select a "None" category if they did not think any would contain relevant links. Each of the four annotators performed exactly the same annotation – it was done four times so agreement could be calculated.

7 Results and Analysis

Our results are calculated on two metrics: relevance and predictivity, as described in the previous section.

7.1 Relevance Results

The results of the evaluation are summarized in Table 2. The table reports four statistics for each of the systems compared. In the table, **MSN** is vanilla MSN search and **QDSE** is the system described in this paper.

The first statistic reported in the table is probability of success using this system (number of successful searches divided by the number of total searches). Next, Avg. Success Cost, is the average cost of the relevant URL for that system. This cost averages only over the successes (queries for which a relevant URL was found). The next statistic, Avg. Cost, is the average cost including failures, where the cost of a failure is, in the case of vanilla MSN, the number of returned results and, in the case of QDSE, the cost of reading the top five results, all the labels and one category expansion[5]. The last statistic, Avg. Mutual

[4] The interface used for evaluation converted the QDSE results into a linear list using our linearization technique so that the interface would be indistinguishable between the two systems.

[5] It is also possible that the user could have guessed looking just at the category expansions that his search had failed, in which case the cost would be less. In this sense, this is an over-estimate of the cost for failed QDSE searches.

	MSN	QDSE
Prob. Success	88.0%	67.7%
Avg. Success Cost	12.4	4.7
Avg. Cost	22.9	9.0
Avg. Mutual Cost	23.0	9.0
kappa	0.57	0.45

Table 2: Results of the evaluation

Cost, is the average cost for all pairs where both systems found a relevant document. The last line reports inter-annotator agreement as calculated over the 12 pairs. This is low, due partly to the small sample size and partly to the fact that the intents themselves were still somewhat underspecified.

7.2 Predictivity Results

We performed two calculations on the results of the predictivity annotations. In the first calculation, we consider the relevance judgments on the QDSE system to be the gold standard. Whenever a relevant link was found in the top 5 (i.e., in the vanilla MSN Search results), this query was thrown out (our model is that the user reads these top five first and only then reads the categories – in the case when one of these top five is relevant, there is no need for him to go further). Excluding these pairs, we calculated average precision of choosing the correct first category. This measures the extent to which the oracle system is correct. On this task, precision was 0.54.

The second calculation we made was to determine whether a user can predict, looking at the headers only, whether their search has been successful. In the task of simply identifying failed searches, precision was 0.70.

Inter-annotator agreement for predictivity was somewhat low, with a kappa value of only 0.49.

7.3 Analysis

As can be seen from Table 2, a user is actually less likely to find a relevant query in the top 100 documents using the QDSE system than using the MSN system. However, this is a somewhat artificial task: experience shows that very few users will actually read through the top 100 returned documents before giving up. Moreover, as seen in the evaluation of the predictivity results, users can decide, with a precision of 0.70, whether their search has failed having read only the category labels. This is in stark contrast to the vanilla MSN search where they could not know without reading all the results whether their search had succeeded.

Moreover, if one does not wish to give up on recall at all, we could simply list all the MSN search results immediately after the QDSE results (removing the top 5, since these will be duplicates). By doing this, we ensure that the probability of success is at least as high for the QDSE system. We can estimate the additional cost this would incur to the QDSE system as follows. First, we notice that Avg. Cost and Avg. Mutual Cost are nearly identical for the MSN system. This means that regardless of whether a query was successful or not for the QDSE system, the cost for a single query in the MSN system is about 23. In particular, this means that the cost for a single query in the MSN system *for which the QDSE system failed* is also about 23. Thus, in the $\frac{88.0-67.7}{88.0} = 23$ percent of the cases where QDSE failed but MSN didn't, we can assume an additional cost for QDSE of $23 - 5 = 18$ (the top 5 are removed). This comes to a total cost of $0.23 * 18 = 4.15$. Thus, in the situation where we insist on not losing recall, the cost of the QDSE+MSN combined system is 13.2, still superior to the vanilla MSN search.

If one is optimistic and is willing to assume that a user will know based only on the category labels whether or not their search has succeeded, then the interesting comparison from Table 2 is between Avg. Success Cost for QDSE and Avg. Cost for MSN. In this case, our cost of 4.7 is a factor of 5 better than the MSN cost. Even if one does not accept our hypothesis that users can predict the correct category, this still leaves room for them to guess incorrectly three times (and hence view four full categories) before our method begins to get more costly than vanilla search.

If, on the other hand, one is pessimistic and believes that a user will not be able to identify based on the category names whether or not their search has succeeded in the QDSE system, then the interesting comparison is between the Avg. Costs for MSN and QDSE. Again, here, we can see that the user can try three categories before vanilla MSN search wins.

One can also be extremely pessimistic and believe that you cannot tell based on the category names in QDSE whether or not you have succeeded, but *can* tell in MSN. It is quite unlikely that this is the case, but even if it is, the interesting comparison is between the Avg. Success Cost for MSN and the Avg. Cost for QDSE. Even in this case, our system still wins out by a cost of 9.0 to 12.4, which is enough for them to guess the wrong category once and then read through half of another category.

8 Conclusion

We have presented a method for providing useful suggested queries for underspecified informational queries by partitioning search results according to related queries.

We evaluated our system using an unbiased metric against a standard web search system and found that, for informational queries, our system enables users to more quickly find their desired information. This conclusion is based on an "oracle" assumption, which we also evaluate. Based on these evaluations, we can show that even under a pessimistic view point, our system outperforms the vanilla search engine.

There is still room for improvement, especially in the predictivity results. We would like users to be able to more readily identify the class into which a relevant document (if one exists) would be found. We are investigating multi-document summarization techniques which might allow users to better pinpoint the category in which a relevant document might be found.

References

Doug Beeferman and Adam Berger. 2000. Agglomerative clustering of a search engine query log. In *Knowledge Discovery and Data Mining*, pages 407–416.

Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117.

A. Broder. 2002. A taxonomy of web search. In *SIGIR Forum*, volume 36(2).

Jaime G. Carbonell and Jade Goldstein. 1998. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Research and Development in Information Retrieval*, pages 335–336.

Susan T. Dumais and Hao Chen. 2000. Hierarchical classification of Web content. In Nicholas J. Belkin, Peter Ingwersen, and Mun-Kew Leong, editors, *Proceedings of SIGIR-00, 23rd ACM International Conference on Research and Development in Information Retrieval*, pages 256–263, Athens, GR. ACM Press, New York, US.

Susan T. Dumais, Edward Cutrell, and Hao Chen. 2001. Optimizing search by showing results in context. In *CHI*, pages 277–284.

Anton Leuski and James Allan. 2000. Details of lighthouse. Technical Report IR-212, Center for Intelligent Information Retrieval.

Oren Zamir and Oren Etzioni. 1999. Grouper: a dynamic clustering interface to Web search results. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1361–1374.