The Hedge Algorithm for Metasearch at TREC 2006

Javed A. Aslam* Virgil Pavlu Carlos Rei
College of Computer and Information Science
Northeastern University
{jaa,vip,crei}@ccs.neu.edu

February 7, 2007

Abstract

Aslam, Pavlu, and Savell [3] introduced the Hedge algorithm for metasearch which effectively *combines* the ranked lists of documents returned by multiple retrieval systems in response to a given query and *learns* which documents are likely to be relevant from a sequence of on-line relevance judgments. It has been demonstrated that the Hedge algorithm is an effective technique for metasearch, often significantly exceeding the performance of standard metasearch and IR techniques over small TREC collections. In this work, we explore the effectiveness of Hedge over the much larger Terabyte 2006 collection.

1 Introduction

Aslam, Pavlu, and Savell introduced a unified framework for simultaneously solving the problems of metasearch, pooling, and system evaluation based on the Hedge algorithm for on-line learning [3]. Given the ranked lists of documents returned by a collection of IR systems in response to a given query, Hedge is capable of matching and often exceeding the performance of the best underlying retrieval system; given relevance feedback, Hedge is capable of "learning" how to optimally combine the input systems, yielding a level of performance which often significantly exceeds that of the best underlying system.

In previous experiments with smaller TREC collections [3], it has been shown that after only a handful of judged feedback documents, Hedge is able to significantly outperform the CombMNZ and Condorcet metasearch techniques. It has also been shown that Hedge is able to efficiently construct pools containing significant numbers of relevant documents and that these pools are highly effective at evaluating the underlying systems [3]. Although the Hedge algorithm has been shown to be a strong technique for metasearch, pooling, and system evaluation using the relatively small or moderate TREC collections (TRECs 3, 5, 6, 7, 8), it has yet to be demonstrated that the technique is scalable to corpora whose data size is at the terabyte level. In this work, we assess the performance of Hedge on a terabyte scale, summarizing training results using the Terabyte 2005 queries and data and presenting testing results using the Terabyte 2006 queries and data.

Finally, we note that in the context of TREC, the Hedge algorithm is both an automatic and a manual technique: In the absence of feedback, Hedge is a fully automatic metasearch algorithm; in the presence of feedback, Hedge is a manual technique, capable of "learning" how to optimally combine the underlying systems.

1.1 Metasearch

The problem of metasearch [2, 7, 10, 9, 11, 12, 4] is to combine the ranked lists of documents output by multiple retrieval systems in response to a given query so as to optimize the quality of the combination and hopefully exceed the performance of the best underlying system. Aslam, Pavlu, and Savell [3] considered two benchmark metasearch techniques for assessing how well their Hedge algorithm performed: (1) CombMNZ, a technique which sums the (appropriately normalized) relevance scores assigned to each document by the underlying retrieval systems and then multiplies that summation by the number of systems that retrieved the document and (2) Condorcet, a technique based on a well known method for conducting a multicandidate election, where the doc-

*We gratefully acknowledge the support provided by NSF grants CCF-0418390 and IIS-0534482.

uments act as candidates and the retrieval systems act as voters providing preferential rankings among these candidates. In experiments using the TREC 3, 5, 6, 7, and 8 collections, Aslam et al. demonstrated that, in the absence of feedback, Hedge consistently outperforms Condorcet and at least equals the performance of CombMNZ; in the presence of even modest amounts of user feedback, Hedge significantly outperforms both CombMNZ and Condorcet, as well as the best underlying system.

In this work, we discuss our experiments with the Hedge algorithm in the Terabyte track at TREC 2006, and we also compare to those results obtained by using the Hedge algorithm run over the data from the Terabyte track at TREC 2005. In the sections that follow, we begin by briefly describing our methodology and experimental setup, and we then describe our results and conclude with future work.

2 Methodology

We implemented and tested the Hedge algorithm for metasearch as described in Aslam et al. [3]. While the details of the Hedge algorithm can be found in the aforementioned paper, the relevant intuition for this technique, as quoted from this paper, is given below.

> Consider a user who submits a given query to multiple search engines and receives a collection of ranked lists in response. How would the user select documents to read in order to satisfy his or her information need? In the absence of any knowledge about the quality of the underlying systems, the user would probably begin by selecting some document which is "highly ranked" by "many" systems; such a document has, in effect, the collective weight of the underlying systems behind it. If the selected document were relevant, the user would begin to "trust" systems which retrieved this document highly (i.e., they would be "rewarded"), while the user would begin to "lose faith" in systems which did not retrieve this document highly (i.e., they would be "punished"). Conversely, if the document were non-relevant, the user would punish systems which retrieved the document highly and reward systems which did not. In subsequent rounds, the user would likely select documents according to his or her faith in the various systems in conjunction with how these systems rank the various documents; in other words, the user would likely pick documents which are *ranked highly* by *trusted* systems.

Our Hedge algorithm for on-line metasearch precisely encodes the above intution using the well studied Hedge algorithm for on-line learning, first proposed by Freund and Schapire [8]. In our generalization of the Hedge algorithm, Hedge assigns a weight to each system corresponding to Hedge's computed "trust" in that system, and each system assigns a weight to each document corresponding to its "trust" in that document; the overall score assigned to a document is the sum, over all systems, of the product of the Hedge weight assigned to the system (a quantity which varies given user feedback) and the system's weight assigned to that document (a fixed quantity which is a function of the rank of that document according to the system). The weights Hedge assigns to systems are initially uniform, and they are updated given user feedback (in line with the intuition given above), and the document set is dynamically ranked according to the overall document scores which change as the Hedge-assigned system weights change.

Initially, Hedge assigns a uniform weight to all systems and computes overall scores for the documents as described above; the ranked list of documents ordered by these scores is created, and we refer to this system and corresponding list as "hedge0." A user would naturally begin by examining the top document in this list, and Hedge would seek feedback on the relevance of that document. Given this feedback, Hedge will assign new system weights (rewarding those systems that performed "well" with respect to this document and punishing those that did not), and it will assign new overall scores to the documents based on these new system weights. The remaining unjudged documents would then be re-ranked according to these updated scores, and this new list would be presented to the user in the next round.

After k documents have been judged, the performance of "hedge k" can be assessed from at least two perspectives, which we refer to as the "user experience" and the "research librarian" perspectives, respectively.

- *User experience:* Concatenate the list of k judged documents (in the order that they were presented to the user) with ranking of the

unjudged documents produced at the end of round k. This concatenated list corresponds to the "user experience," i.e., the ordered documents that have been examined so far along with those that will be examined if no further feedback is provided.

- *Research librarian:* Concatenate the *relevant* subset of the k judged documents with the ranking of the unjudged documents produced at the end of round k. This concatenated list corresponds to what a research librarian using the Hedge system might present to a client: the relevant documents found thus far followed by the ordered list of unjudged documents in the collection.

Note that the performance of the "research librarian" is likely to exceed that of the "user experience" by any reasonable measure of retrieval performance since judged non-relevant documents are eliminated from the former concatenated list. In what follows, "hedge k" refers to the system, concatenated list, and performance as defined with respect to the "research librarian" perspective.

3 Experimental Setup and Results

We tested the performance of the Hedge algorithm by using the queries from TREC 2005 Terabyte Track. Then we run Hedge for Terabyte06 track, using real user feedback (we judged 50 documents per query). Both Terabyte05 and Terabyte06 use the GOV2 collection of about 25 million documents. We indexed the collection using the Lemur Toolkit; that process took about 3 days using a 2-processor dual-core Opteron machine (2.4 GHz/core).

3.1 Underlying IR systems

The underlying systems include: (1) two tf-idf retrieval systems; (2) three KL-divergence retrieval models, one with Dirichlet prior smoothing, one with Jelinek-Mercer smoothing, and the last with absolute discounting; (3) a cosine similarity model; (4) the OKAPI retrieval model; (5) and the INQUERY retrieval method. All of the above retrieval models are provided as standard IR systems by the Lemur Toolkit [1].

These models were run against a collection (GOV2) of web data crawled from web sites in the .gov domain during early 2004 by NIST [6]. The collection is 426GB in size and contains 25 million documents [6]. Although this collection is not a full terabyte in size, it is still much larger than the collections used at previous TREC conferences.

For each query and retrieval system, we considered the top 10,000 scored documents for that retrieval system. Once all retrieval systems were run against all queries, we ran the Hedge algorithm described above to perform metasearch on the ranked lists we obtained.

3.2 Results using Terabyte 2005 queries and qrel

We used the TREC 2005 qrel files to provide Hedge with relevance feedback. If one of our underlying systems retrieved a document that was not included in the qrel file, we assumed the document to be non-relevant.

Hedge was run as follows. In the first round each of the underlying systems all have an equal weight and the underlying lists are fused by ranking documents according to highest weighted average mixture loss [3]. The initial run of Hedge (hedge0) will not acquire any relevance judgments and hence can be compared directly to standard metasearch techniques [3] (e.g. CombMNZ).

In the following round, the top document from hedge0 is judged. In our case, we obtain the judgment from TREC qrel file (0 if document not in the qrel). If the document is relevant, it is put at the top our metasearch list, and if it is not, it is discarded. The judgment is then used to re-weight the underlying systems. As described above, systems are re-weigthed based on the rank of the document just judged. Then a new metasearch list is produced, corresponding to hedge1. The next round proceeds in the same manner: the top unjudged document from the last metasearch list is judged and then used to: (1) identify where the document should be placed in the list; (2) update the system weight vector to reward the correct systems and punish the incorrect systems; (3) re-rank the remaining unjudged documents.

In our experiments we had 50 rounds (relevance judgments) and we note the results of hedge for 0, 5, 10, 15, 20, 30, and 50 judgments.

For comparison, we also ran Condorcet and CombMNZ over the ranked lists generated by our underlying systems. We then calculated mean average precision scores for each of the three metasearch systems and compared the performance of the Hedge system with the performance of the lists generated by Condorcet and CombMNZ (see Figure 1.).

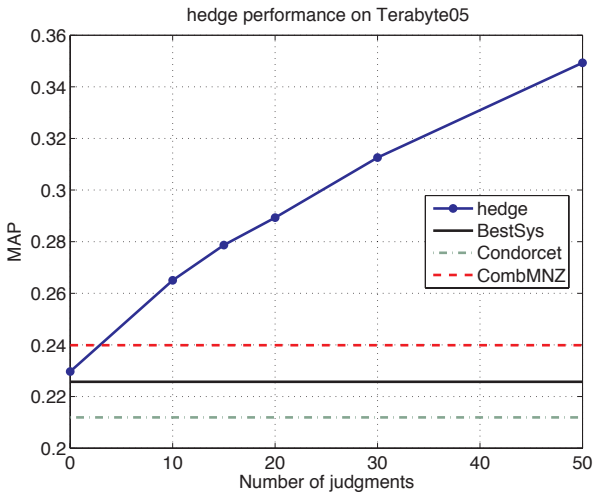

Figure 1: Terabyte05: Hedge-m: metasearch performance as more documents are judged.

We compare Hedge to CombMNZ, Condorcet, and the underlying retrieval systems that were used for our metasearch technique. Table 2 shows that Hedge, in the absence of any relevance feedback (hedge0), consistently outperforms Condorcet. The performance of hedge0 is comparable with the performance of CombMNZ.

Table 2 illustrates that both hedge0 and CombMNZ are able to exceed the performance of the best underlying system. This demonstrates that Hedge alone, even without any relevance feedback, is a successful metasearch technique.

After providing the Hedge algorithm with only ten relevance judgments (hedge10), Hedge significantly outperforms CombMNZ, Condorcet, and the best underlying system in terms of MAP (Table 1). Also hedge50 more than doubles precision at cutoff 20 of the top underlying system. This is in part because

System	2	4	6	8
CombMNZ	0.2332	0.2693	0.2715	0.2399
Condorcet	0.1997	0.2264	0.2302	0.2119
Hedge 0	0.2314	0.2641	0.2687	0.2297
Hedge 10	0.2579	0.2944	0.2991	0.2650
Hedge 50	0.3199	0.3669	0.3652	0.3493

Table 1: Terabyte05: Hedge vs. Metasearch Techniques CombMNZ and Condorcet, combining 2 , 4, 6, 8 underlying systems.

System	*MAP*	*p@20*
Jelinek-Mercer	0.2257	0.3780
Dirichlet	0.2100	0.4200
TFIDF	0.1993	0.4250
Okapi	0.1906	0.4270
log-TFIDF	0.1661	0.4140
Absolute Discounting	0.1575	0.3660
Cosine Similarity	0.0875	0.1960
CombMNZ	0.2399	0.4550
Condorcet	0.2119	0.4200
hedge0	0.2297	0.4260
hedge10	0.2650	0.5270
hedge50	0.3493	0.8090

Table 2: Results for input and metasearch systems on Terabyte05. CombMNZ, Cordorcet, and Hedge N were run over all input systems.

documents that have been ranked relevant are placed at the top of the list, whereas the documents that have been judged non-relevant are discarded.

3.3 Results for Terabyte 2006 queries

For our Terabyte submission to TREC 2006, given the lack of judgments, we manually judged several documents for each query. We choose to run Hedge for 50 rounds (for each query) on top of our underlying IR systems (provided by Lemur, as described above). Therefore, in total, 50 rounds x 50 queries = 2500 documents were judged for relevance.

As a function of the amount of relevance feedback utilised, four different runs were submitted to Terabyte 2006: hedge0 (no judgments), which is essentially an automatic metasearch system; hedge10 (10 judgments per query); hedge30 (30 judgments per query) and hedge50 (50 judgments per query). The performance of all four runs are presented in Table 3.

The table reports the mean average precision (MAP), R-precison, and precision-at-cutoff 10, 30, 100 and 500. Against expectations, hedge30 looks slightly better than hedge50 but this is most likely due to the fact that hedge30 was included as a contributor to the TREC pool of judged documents while hedge50 was not.

3.4 Judgment disagreement and impact to Hedge performance

Hedge works as an on-line metasearch algorithm, using user feedback (judged documents) to weight underlying input systems. It does not have a "search engine" component; i.e., it does not perform traditional retrieval by analyzing documents for relevance to a given query. Therefore the performance is heavily determined by user feedback, i.e., the quality of he judgments. In what follows, we discuss how well our own judgments (50 per query) match those provided by TREC qrel file, released at the conclusion of TREC 2006. Major disagreements could obviously lead to significant changes in performance. First, we note that there are consistent, large disagreements. Mismatched relevance judgments for Query 823 are shown below:

GX000-62-7241305	trecrel=0	hedgerel=1
GX000-14-5445022	trecrel=1	hedgerel=0
GX240-72-4498727	trecrel=1	hedgerel=0
GX060-85-9197519	ABSENT	hedgerel=0
GX240-48-7256267	trecrel=1	hedgerel=0
GX248-73-4320232	trecrel=1	hedgerel=0
GX245-68-14099084	trecrel=0	hedgerel=1
GX227-60-13210050	trecrel=1	hedgerel=0
GX071-71-15063229	trecrel=1	hedgerel=0
GX047-80-14304963	trecrel=1	hedgerel=0
GX217-86-0259964	trecrel=1	hedgerel=0
GX031-42-14513498	trecrel=1	hedgerel=0
GX227-75-10978947	trecrel=1	hedgerel=0
GX004-97-14821140	trecrel=1	hedgerel=0
GX268-65-3825487	ABSENT	hedgerel=0
GX029-22-6233173	trecrel=1	hedgerel=0
GX060-96-11856158	ABSENT	hedgerel=0
GX269-71-3058600	trecrel=1	hedgerel=0
GX271-79-2767287	trecrel=1	hedgerel=0
823	19 mismatches	

We examined a subset of the mismatched relevance judgments and we believe that there were judgment errors on both sides. Nevertheless all judgment disagreements on judges affect *measured* hedge performance negatively. For comparison we re-run hedge30 (30 judgments) using the TREC qrel file for relevance feedback. In doing so, we obtained a mean average precision of 0.33, consistent with performance on Terabyte 2005. This would place the new hedge30 run second among all manual runs, as ordered by MAP (Figure 2).

We also looked at this new run (hedge30 with TREC qrel file instead of user feedback) on a query-by-query basis. Figure 3 shows a scatterplot comparison, per query, of the original hedge30 performance and the performance using TREC judgments for feedback. Note the significant and nearly uniform improvements obtained using TREC judgments.

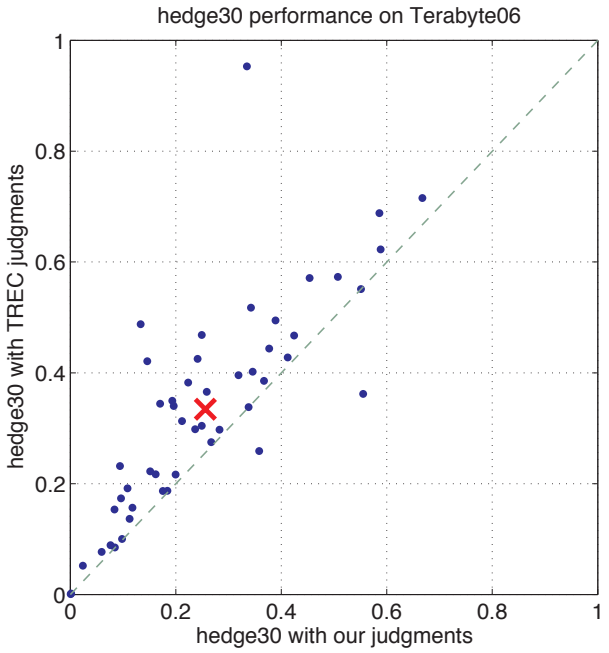

Figure 3: Terabyte06: hedge30. Each dot corresponds to a query; x-axis corresponds to hedge30 AP values obtained with our judgments as user feedback; y-axis corresponds to hedge30 AP values using TREC qrel file for feedback. MAP vaues are denoted by "×".

4 Conclusions

It has been shown that the Hedge algorithm for online learning is highly efficient and effective as a metasearch technique. Our experiments show that even without relevance feedback Hedge is still able to produce metasearch lists which are directly comparable to the standard metasearch techniques Condorcet and CombMNZ, and which exceed the performance of the best underlying list. With relevance feedback

System	*MAP*	*R-prec*	*p@10*	*p@30*	*p@100*	*p@500*
hedge0	0.177	0.228	0.378	0.320	0.232	0.104
hedge10	0.239	0.282	0.522	0.394	0.278	0.118
hedge30	0.256	0.286	0.646	0.451	0.290	0.119
hedge50	0.250	0.280	0.682	0.470	0.279	0.115

Table 3: Results for Hedge runs on Terabyte06 queries.

Group	**Run**	**bpref**	**p@20**	**MAP**
uwaterloo-clarke	uwmtFmanual	0.4785	0.7030	0.4246
sabir.buckley	sabtb06man1	0.4104	0.6070	0.2666
pekingu.yan	TWTB06AD02	0.4089	0.5070	0.3152
rmit.scholer	zetaman	0.3976	0.5290	0.2873
umilano.vigna	mg4jAdhocBV	0.3944	0.4930	0.2822
umelbourne.ngoc-anh	MU06TBa1	0.3900	0.5420	0.2927
ecole-des-mines.beigbeder	AMRIMtpm5006	0.3793	0.4390	0.2705
ibm.carmel	JuruMan	0.3570	0.5190	0.2754
northeasternu.aslam	hedge30	0.3180	0.5110	0.2561
max-planck.theobald	mpiirmanual	0.3041	0.4810	0.1981
ualaska.fairbanks.newby	arscDomManL	0.1202	0.0400	0.0351

```
vip@fiji1 | ~/crei/lemur/Queries/T06/Results_final
18:32>> trec_eval qrels.tb06.top50 hedge30

Queryid (Num):      50
Total number of documents over all queries
    Retrieved:   500000
    Relevant:      5893
    Rel_ret:       5052
Interpolated Recall - Precision Averages:
    at 0.00       0.9554
    at 0.10       0.8101
    at 0.20       0.6687
    at 0.30       0.4189
    at 0.40       0.3391
    at 0.50       0.2600
    at 0.60       0.2034
    at 0.70       0.1458
    at 0.80       0.1012
    at 0.90       0.0619
    at 1.00       0.0117
Average precision (non-interpolated) for all rel docs
                  0.3343

Precision:
  At    5 docs:   0.8960
  At   10 docs:   0.8140
  At   15 docs:   0.7293
  At   20 docs:   0.6610
  At   30 docs:   0.5507
  At  100 docs:   0.3128
  At  200 docs:   0.2226
  At  500 docs:   0.1234
  At 1000 docs:   0.0737
R-Precision (precision after R (= num_rel for a query
    Exact:        0.3371
```

Figure 2: Terabyte06: hedge30 with TREC qrel judgments. The shell shows trec_eval measurements on top of the published TREC Terabyte06 ranking of manual runs [5]; It would rank second in terms of MAP.

Hedge is able to considerably outperform Condorcet and CombMNZ.

The performance shown when using TREC qrels file was consistently very good; when using our judgments the relatively poor performance was due to using a set of judgments for feedback and a different set of judgments for evaluation. Ultimately we believe that Hedge is somehow immune to judge disagreement, as long us the feedback comes from the same source (or judge or user) as the performance measurement. Certainly, in practice, it is possible that two users ask the same query but they are looking for different information; in this case user feedback would be different which would lead to different metasearch lists produced and eventually to a satisfactory performance for each user.

References

[1] The lemur toolkit for language modeling and information retrieval. http://www.cs.cmu.edu/~lemur.

[2] Javed A. Aslam and Mark Montague. Models for metasearch. In W. Bruce Croft, David J. Harper, Donald H. Kraft, and Justin Zobel, editors, *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 276–284. ACM Press, September 2001.

[3] Javed A. Aslam, Virgiliu Pavlu, and Robert Savell. A unified model for metasearch, pooling, and system evaluation. In Ophir Frieder, Joachim Hammer, Sajda Quershi, and Len Seligman, editors, *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, pages 484–491. ACM Press, November 2003.

[4] Brian T. Bartell, Garrison W. Cottrell, and Richard K. Belew. Automatic combination of multiple ranked retrieval systems. In *SIGIR 94*, pages 173–181.

[5] Stefan Büttcher, Charles L. A. Clarke, and Ian Soboroff. The TREC 2006 terabyte track. In

Proceedings of the Fifteen Text REtrieval Conference (TREC 2006), 2006.

[6] Charles L. A. Clarke, Falk Scholer, and Ian Soboroff. The TREC 2005 terabyte track. In *Proceedings of the Fourteenth Text REtrieval Conference (TREC 2005)*, 2005.

[7] Edward A. Fox and Joseph A. Shaw. Combination of multiple searches. In *TREC 94*, pages 243–249.

[8] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

[9] Joon Ho Lee. Analyses of multiple evidence combination. In *SIGIR 97*, pages 267–275.

[10] Joon Ho Lee. Combining multiple evidence from different properties of weighting schemes. In *SIGIR 95*, pages 180–188.

[11] R. Manmatha, T. Rath, and F. Feng. Modeling score distributions for combining the outputs of search engines. In *SIGIR 2001*, pages 267–275.

[12] Christopher C. Vogt. How much more is better? Characterizing the effects of adding more IR systems to a combination. In *RIAO 2000*, pages 457–475.