A Unified Model for Metasearch, Pooling, and System Evaluation*

Javed A. Aslam
College of Computer Science
Northeastern University
360 Huntington Ave, #161CN
Boston, MA 02115
jaa@ccs.neu.edu

Virgiliu Pavlu
College of Computer Science
Northeastern University
360 Huntington Ave, #161CN
Boston, MA 02115
vip@ccs.neu.edu

Robert Savell
Dept. of Computer Science
Dartmouth College
6211 Sudikoff Laboratory
Hanover, NH 03755
rsavell@cs.dartmouth.edu

ABSTRACT

We present a unified model which, given the ranked lists of documents returned by multiple retrieval systems in response to a given query, simultaneously solves the problems of (1) fusing the ranked lists of documents in order to obtain a high-quality combined list (metasearch); (2) generating document collections likely to contain large fractions of relevant documents (pooling); and (3) accurately evaluating the underlying retrieval systems with small numbers of relevance judgments (efficient system assessment). Our approach is based on the Hedge algorithm for on-line learning. In effect, our proposed system "learns" which documents are likely to be relevant from a sequence of on-line relevance judgments. In experiments using TREC data, our methodology is shown to outperform standard methods for metasearch, pooling, and system evaluation, often remarkably so.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Retrieval Models, Search Process, Relevance Feedback

General Terms

Theory, Algorithms, Experimentation

Keywords

Metasearch, Pooling, Evaluation, Active Learning

1. INTRODUCTION

We consider the problems of metasearch, pooling, and system evaluation, and we show that all three problems can be efficiently and effectively solved with a single technique based on the Hedge algorithm for on-line learning. Our results from experiments with TREC data demonstrate that: (1) As an algorithm for metasearch, our technique combines ranked lists of documents in a manner whose performance equals or exceeds that of benchmark algorithms such as CombMNZ and Condorcet, and it generalizes these algorithms by seamlessly incorporating user feedback in order to obtain dramatically improved performance. (2) As an algorithm for pooling, our technique generates sets of documents containing far more relevant documents than standard techniques such as TREC-style depth pooling. (3) These pools, when used to evaluate retrieval systems, estimate the performance of retrieval systems and rank these systems in a manner superior to TREC-style depth pools of an equivalent size.

Our unified model for solving these three problems is based on the Hedge algorithm for on-line learning. In the context of these problems, Hedge effectively learns which systems are "better" than others and which documents are "more likely relevant" than others, given on-line relevance feedback. Thus, Hedge (1) learns to rank documents in order of relevance (metasearch), (2) learns how to generate document sets likely to contain large fractions of relevant documents (pooling), and (3) efficiently and effectively evaluates the underlying retrieval systems using these pools.

In the sections that follow, we describe the problems of metasearch, pooling, and system evaluation in more detail and discuss our results.

1.1 Metasearch

Metasearch is the well-studied process of fusing the ranked lists of documents returned by a collection of systems in response to a given user query in order to obtain a combined list whose quality equals or exceeds that of any of the underlying lists. Many metasearch techniques have been proposed and studied [3, 9, 14, 2, 12]. In this work, we consider two benchmark techniques: the first is based on combining the normalized scores given to each document by the underlying systems (CombMNZ [6, 10]), and the second is based on viewing the metasearch problem as a multi-candidate election where the documents are candidates and the systems are voters expressing preferential rankings among the candidates (Condorcet [13]).

CombMNZ and Condorcet produce quality ranked lists of documents by fusing the ranked lists provided by a col-

*This work partially supported by NSF Career Grant CCR-0093131 and NSF Grant 5-36955.

CIKM'03, November 3–8, 2003, New Orleans, Louisiana, USA.

lection of underlying systems. Given ranked lists produced by good but sufficiently different underlying systems, these metasearch techniques can produce fused lists whose quality exceeds that of any of the underlying lists. Given ranked lists produced by possibly correlated systems of varying performance, these metasearch techniques will most often produce fused lists whose performance exceeds that of the "average" underlying list but which rarely exceeds that of the best underlying list.

In the context of a metasearch engine, the fused list produced by CombMNZ or Condorcet would be presented to the user who would naturally begin processing the documents in rank order to satisfy the desired information need. While the user could naturally and easily provide relevance feedback to the metasearch algorithm, these techniques are not naturally amenable to incorporating such feedback.

By contrast, our technique based on the Hedge algorithm for on-line learning quite naturally incorporates relevance feedback and performs impressively even in the absence of feedback.

In the absence of feedback, the metasearch performance of our technique most often equals or exceeds that of benchmark techniques such as CombMNZ and Condorcet. In experiments using TREC data, Hedge effectively equaled the performance of CombMNZ in four out of five data sets tested (TRECs 3, 6, 7, and 8), and Hedge outperformed CombMNZ in one data set (TREC 5). Hedge consistently outperformed Condorcet in each of the data sets tested, significantly so in two of them (TRECs 6 and 7).

In the presence of relevance feedback, our technique rapidly and effectively "learns" how to fuse the underlying ranked lists, significantly outperforming CombMNZ and Condorcet, and often outperforming the best underlying system after only a handful of relevance judgments.

1.2 Pooling and System Evaluation

Collections of retrieval systems are traditionally evaluated by (1) constructing a test collection of documents (the "corpus"), (2) constructing a test collection of queries (the "topics"), (3) judging the relevance of the documents to each query (the "relevance judgments"), and (4) assessing the quality of the ranked lists of documents returned by each retrieval system for each topic using standard measures of performance such as mean average precision. Much thought and research has been devoted to each of these steps in, for example, the annual TREC conference [8].

For large collections of documents and/or topics, it is impractical to assess the relevance of each document to each topic. Instead, a small subset of the documents is chosen, and the relevance of these documents to the topics is assessed. When evaluating the performance of a collection of retrieval systems, as in the annual TREC conference [8], this judged "pool" of documents is typically constructed by taking the union of the top k documents returned by each system in response to a given query. In TREC, $k = 100$ has been shown to be an effective cutoff in evaluating the relative performance of retrieval systems [8]. Both shallower and deeper pools have been studied [16, 8], both for TREC and within the greater context of the generation of large test collections [4]. Pooling is an effective technique since many of the documents relevant to a topic will appear near the top of the lists returned by (quality) retrieval systems; thus, these relevant documents will be judged and used to effectively assess the performance of the collected systems.

While pooling is an effective technique for greatly reducing the number of relevance judgments required for effective system evaluation, it can still be quite expensive. In the TREC conference, for example, upwards of 100 systems return lists of 1000 ranked documents in response to each of 50 topics. Traditional TREC-style pooling dictates that the top 100 documents returned by each system in response to each topic should be judged, and these relevance judgments should then be used to assess the relative performance of the systems. While many of the top documents are retrieved by multiple systems, thus reducing the overall size of the pool, the total number of relevance judgments is still substantial. For example, in TREC 8 [15] 86,830 relevance judgments were used to assess the quality of the retrieved lists submitted by 129 systems in response to 50 topics.

Pools are often used to evaluate retrieval systems in the following manner. The documents within a pool are judged to determine whether they are relevant or not relevant to the given user query or topic. Documents not contained within the pool are *assumed* to be non-relevant. The ranked lists returned by the retrieval systems are then evaluated using standard measures of performance (such as mean average precision) using this "complete" set of relevance judgments. Since documents not present in the pool are assumed non-relevant, the *quality* of the assessments produced by such a pool is often in direct proportion to the fraction of relevant documents found in the pool (its *recall*). On-line pooling techniques have been proposed which attempt to identify relevant documents as quickly as possible in order to exploit this phenomenon [4].

While TREC-style pools with large numbers of relevance judgments corresponding to documents chosen "fairly" from

	TREC				
Pool	3	5	6	7	8
Depth	$n = 40$	$n = 82$	$n = 79$	$n = 103$	$n = 129$
1	19	38	38	32	40
2	39	68	67	55	69
3	47	98	95	76	95
4	60	126	120	95	119
5	73	153	146	114	144
6	85	181	172	134	167
7	96	208	197	152	191
8	107	234	221	170	215
9	118	262	246	189	238
10	129	288	271	207	260
15	183	418	393	297	379
20	235	543	513	389	494
30	336	791	743	571	717
40	436	1034	969	754	939
50	531	1273	1191	936	1155
60	626	1509	1410	1114	1366
70	718	1745	1629	1299	1574
80	811	1978	1845	1486	1777
90	903	2206	2058	1675	1978
100	995	2434	2271	1860	2176

Table 1: The size of the pool (per query) for various pool depths if the pooling is performed TREC-style. Here n is the number of input systems in the given data set.

among the underlying systems are extremely useful from a research perspective, there are circumstances under which smaller, perhaps biased, pools are warranted. Examples of this might include: (1) when attempting to assess large numbers of systems over vast, changing data collections such as the World Wide Web, (2) when budget or manpower constraints dictate smaller pools, and (3) when the judgments are provided on-line by a user, for example when attempting to quickly determine the best underlying search engine for a particular user-given query.

In the results that follow, we demonstrate that the Hedge algorithm for on-line learning is ideally suited to generating efficient pools which effectively evaluate retrieval systems. In effect, the Hedge algorithm learns which documents are likely to be relevant. These documents can then be judged and added to the pool, and their relevance judgments can be used as feedback to improve the learning process—thus generating more relevant documents in subsequent rounds. The quality of the pools generated can be measured in two ways: (1) At what rate are relevant documents found (recall percentage as a function of total judgments)? (2) How well do these pools evaluate the retrieval systems (score or rank correlations vs. "ground truth")? In our experiments using TREC data, Hedge found relevant documents at rates *nearly double* that of benchmark techniques such as TREC-style depth pooling. These Hedge pools were found to evaluate the underlying retrieval systems much better than TREC-style depth pools of an equivalent size (as measured by Kendall's τ rank correlation, for example). Finally, these Hedge pools seemed particular effective at properly evaluating the best underlying systems, a task which is difficult to achieve using small pools.

In the sections that follow, we first describe our algorithm for simultaneously solving the metasearch, pooling and system evaluation problems using Hedge. We then describe the results of our methodology in experiments conducted on TREC data. Finally, we conclude by mentioning some possible extensions of this work.

2. INTUITION AND SETUP

The intuition for our algorithm can be described as follows. 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.

Algorithm Hedge(β)

Parameters:

number of systems N.
initial weight vector $\mathbf{w}^1 \in [0,1]^N$
number of trials T.
$\beta \in [0,1]$.

Do for $t = 1, 2, \ldots, T$.

1. **Choose allocation $\mathbf{p}^t = \frac{\mathbf{w}^t}{\sum_{i=1}^N w_i^t}$.**
2. **Receive loss $\ell^t \in [0,1]^N$ from environment.**
3. **Suffer loss $\mathbf{p}^t \cdot \ell^t$.**
4. **Set the new weight vector to be $w_i^{t+1} = w_i^t \beta^{\ell_i^t}$.**

Figure 1: Hedge Algorithm

How can the above intuition be quantified and encoded algorithmically? Such questions have been studied in the machine learning community for quite some time and are often referred to as "combination of expert advice" problems. One of the seminal results in this field is the Weighted Majority Algorithm due to Littlestone and Warmuth [11]; in this work, we use a generalization of the Weighted Majority Algorithm called **Hedge** due to Freund and Schapire [7].

Hedge is an on-line allocation strategy which solves the combination of expert advice problem as follows. (See Figure 1.) **Hedge** is parameterized by a tunable learning rate parameter $\beta \in [0,1]$, and in the absence of any *a priori* knowledge, begins with an initially uniform "weight" w_i^1 for each expert i (in our case, $w_i^1 = 1 \; \forall \; i$). The relative weight associated with an expert corresponds to one's "faith" in its performance.

For each round $t \in \{1, \ldots, T\}$, these weights are normalized to form a probability distribution $\mathbf{p}^t$ where

$$\mathbf{p}_i^t = \frac{w_i^t}{\sum_j w_j^t},$$

and one places $\mathbf{p}_i^t$ "faith" in system i during round t.

This "faith" can be manifested in any number of ways, depending on the problem being solved. If the underlying experts are making predictions about which stocks will rise in the next trading day, one might invest one's money in stocks according to the weighted predictions of the underlying experts. If a stock goes up, then each underlying expert i which predicted this rise would receive a "gain," and the investor would also receive a gain in proportion to the money invested, p_i^t. If the stock goes down, then each underlying expert i which predicted a rise would suffer a "loss," and the investor would also suffer a loss in proportion to the money invested. This is encoded in **Hedge** as follows. In each round t, expert i suffers a *loss* ℓ_i^t, and the algorithm suffers a weighted average (*mixture*) loss of $\sum_i p_i^t \ell_i^t$.[1]

[1]For the purposes of the Hedge algorithm and its analysis, it is assumed that the losses and/or gains are bounded so that they can be appropriately mapped to the range $[0,1]$.

Finally, the **Hedge** algorithm updates its "faith" in each expert according to the losses suffered in the current round, $w_i^{t+1} = w_i^t \beta^{\ell_i^t}$. Thus, the greater the loss an expert suffers in round t, the lower its weight in round $t+1$, and the "rate" at which this change occurs is dictated by the tunable parameter β.

Over time, the "best" underlying experts will get the "highest" weights, and the cumulative (mixture) loss suffered by **Hedge** will be not much higher than that of the best underlying expert. Specifically, Freund and Schapire show that if $L_i = \sum_t \ell_i^t$ is the cumulative loss suffered by expert i, then the cumulative (mixture) loss suffered by **Hedge** is bounded by

$$L_{\textbf{Hedge}} \leq \frac{\min_i\{L_i\} \cdot \ln(1/\beta) + \ln N}{1-\beta}$$

where N is the number of underlying experts.

2.1 Hedge Application

We employ the **Hedge** algorithm to simultaneously solve the problems of metasearch, pooling, and system evaluation as follows. On a per query basis, each underlying retrieval system is an "expert" providing "advice" about the relevance of various documents to the given query. We must define a method for selecting likely relevant documents based on system weights and document ranks, and we must also define an appropriate loss that a system should suffer for retrieving a particular relevant or non-relevant document at a specified rank. While a loss function which converges to some standard measure of performance such as average precision might be desirable, we instead work with a simpler but related loss. The loss function is designed to reflect a document's complete contribution to a system's *total precision*—the sum of the precisions at all document levels. It is defined for document d_k at rank r_k by: $\ell = \frac{1}{2} \cdot (-1)^{\mathrm{rel}(d_k)} \cdot \sum_{r=r_k}^{r_{\max}} \frac{1}{r}$, where $\mathrm{rel}(d_k)$ is an indicator function for the relevance of document d_k (i.e., 1 if d_k is relevant and 0 if it is not) and $r_{\max}$ is the total number of unique documents retrieved by all systems for this query (i.e., the size of the union of the documents sets returned). In the limit of complete relevance judgments, one can show that the total loss of a system converges to the negative of the *total precision* plus a system-independent constant. For our purposes, this measure demonstrates a close empirical relationship to other popular measures of performance (such as average precision at relevant documents) while it has the advantage of being simple and "symmetric" (the magnitude of the loss or gain is independent of relevance). We note that this loss can be easily approximated since its magnitude is the difference between two harmonic numbers. Let $H_k = \sum_{i=1}^k 1/i$ be the k-th harmonic number. We then have

$$\begin{aligned}
\ell &= \frac{1}{2} \cdot (-1)^{\mathrm{rel}(d_k)} \cdot \sum_{r=r_k}^{r_{\max}} \frac{1}{r} \\
&= \frac{1}{2} \cdot (-1)^{\mathrm{rel}(d_k)} \cdot \left(\sum_{r=1}^{r_{\max}} \frac{1}{r} - \sum_{r=1}^{r_k - 1} \frac{1}{r} \right) \\
&= \frac{1}{2} \cdot (-1)^{\mathrm{rel}(d_k)} \cdot (H_{r_{\max}} - H_{r_k - 1}) \\
&\approx \frac{1}{2} \cdot (-1)^{\mathrm{rel}(d_k)} \cdot (\ln r_{\max} - \ln(r_k - 1)) \\
&\approx \frac{1}{2} \cdot (-1)^{\mathrm{rel}(d_k)} \cdot \ln \frac{r_{\max}}{r_k}.
\end{aligned}$$

Note that the magnitude of this loss (or gain) is highest for documents which are highly ranked; this is as desired and expected.[2]

Given this loss function, we implement a simple pooling strategy designed to maximize the learning rate of the **Hedge** algorithm. At each iteration, we select the unlabeled document which would maximize the weighted average (mixture) loss if it were non-relevant. Since the loss suffered by a system would be large for a non-relevant document which is highly ranked, this is exactly the unlabeled document with the maximum expectation of relevance as voted by a weighted linear combination of the systems. The strategy is also appropriate for selecting documents to be output in a metasearch list: instead of returning just the single unlabeled document with highest mixture loss if it were non-relevant, rank all the unlabeled documents by their mixture loss if non-relevant, and output this list. (In fact, this list is appended as a suffix to the ordered list of documents that the user has already judged at this point.)

3. EXPERIMENTAL SETUP AND RESULTS

We tested the performance of the above algorithm using data from TRECs 3, 5, 6, 7, and 8. For any given TREC and any given query within that TREC, the experiment proceeds in rounds. In Round 0, each underlying search engine is given an equal Hedge weight, and the corresponding ranked lists are combined by ranking documents according to highest weighted average mixture loss, as described above.[3] This metasearch list created in the absence of relevance judgments, referred to as Hedge-0, can be directly compared to benchmark techniques such as CombMNZ and Condorcet.

In Round 1, the top document from the previous metasearch list is added to an initially empty pool and judged.[4] This judgment can then be used as feedback to reweight the systems and rerank the remaining unlabeled documents

[2]We use r_k in the denominator of the natural log as it avoids a divide-by-zero condition for rank 1 documents. Furthermore, for the purposes of the Hedge algorithm, these losses or gains are mapped to the range $[0, 1]$ by an appropriate shift and scale.

[3]One implementation detail is how to assign a loss to a document which is not retrieved by a particular system. For example, if System i retrieves 1,000 documents and there are $r_{\max}$ total unique documents retrieved by all systems for this query, then there are $r_{\max} - 1000$ unretrieved documents for which losses must be assigned for this system. Each of these documents is given a loss corresponding to the *average* of the losses for documents ranked 1,001 to $r_{\max}$.

[4]Judgments are obtained from the appropriate TREC qrel file; documents unjudged by TREC are assumed non-relevant.

TREC	MNZ	COND	Hedge-0	$\%MNZ$	$\%COND$
3	0.423	0.403	0.418	−1.2	+3.7
5	0.294	0.307	0.309	+5.1	+0.6
6	0.341	0.315	0.345	+1.2	+9.5
7	0.320	0.308	0.323	+0.9	+4.9
8	0.350	0.343	0.352	+1.4	+2.6

Table 2: Hedge-0 Method vs. Metasearch Techniques CombMNZ and Condorcet.

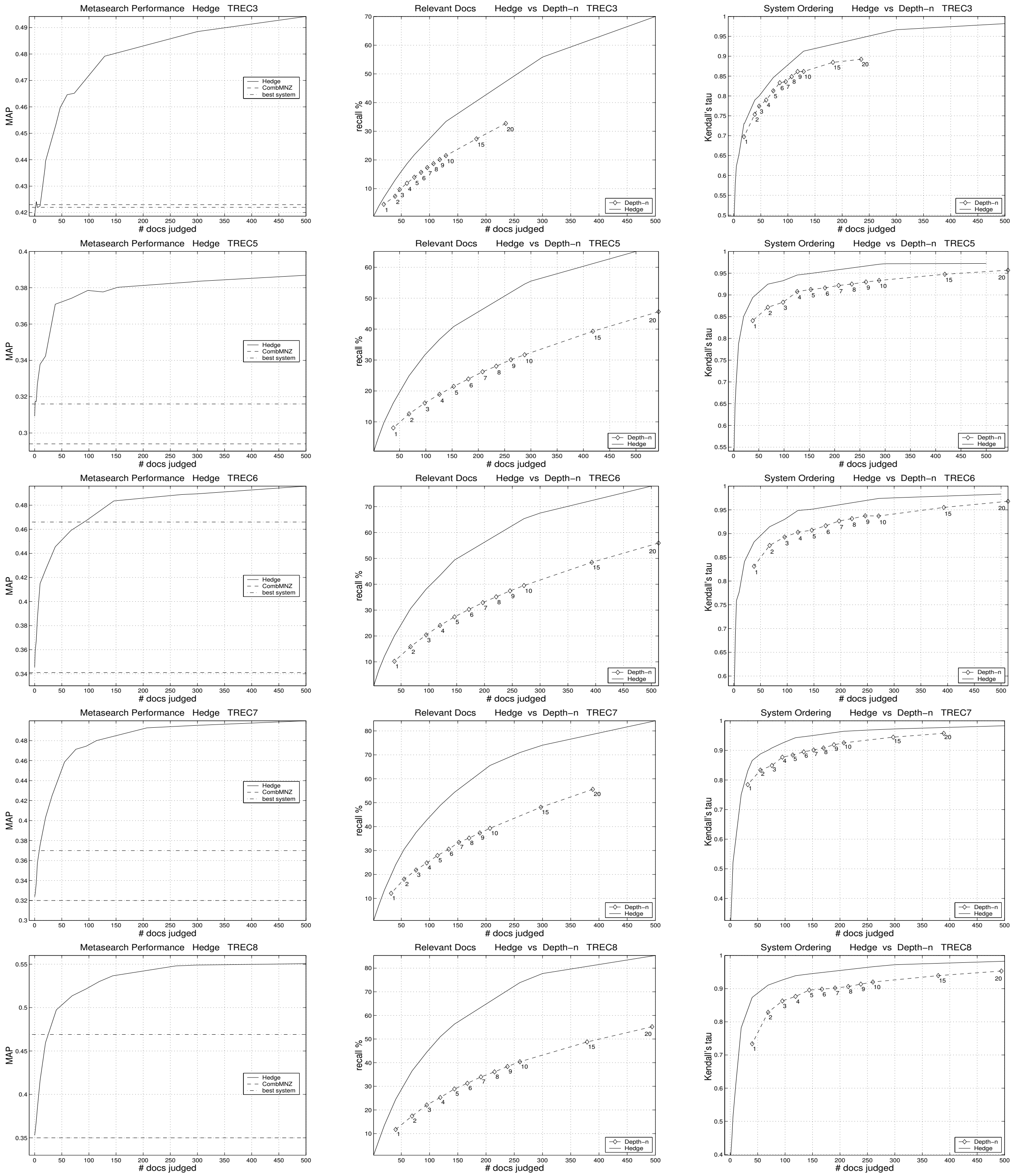

Figure 2: (a)Hedge-m: metasearch performance. (b)Hedge-m vs. Depth-n: percent of total relevant documents discovered. (c) Hedge-m and Depth-n vs. actual ranks: k-τ.

as described above. Thus, the metasearch list in Round 1 corresponds to the judged document (whether relevant or not) followed by the remaining unlabeled documents ranked according to Hedge.

Subsequent rounds proceed in an identical manner: the top *unlabeled* document from the previous metasearch list is judged and added to the pool while the systems are reweighted and the remaining unlabeled documents are reranked by Hedge given this feedback. Thus, the pool in Round j will consist of the j documents judged by the user,

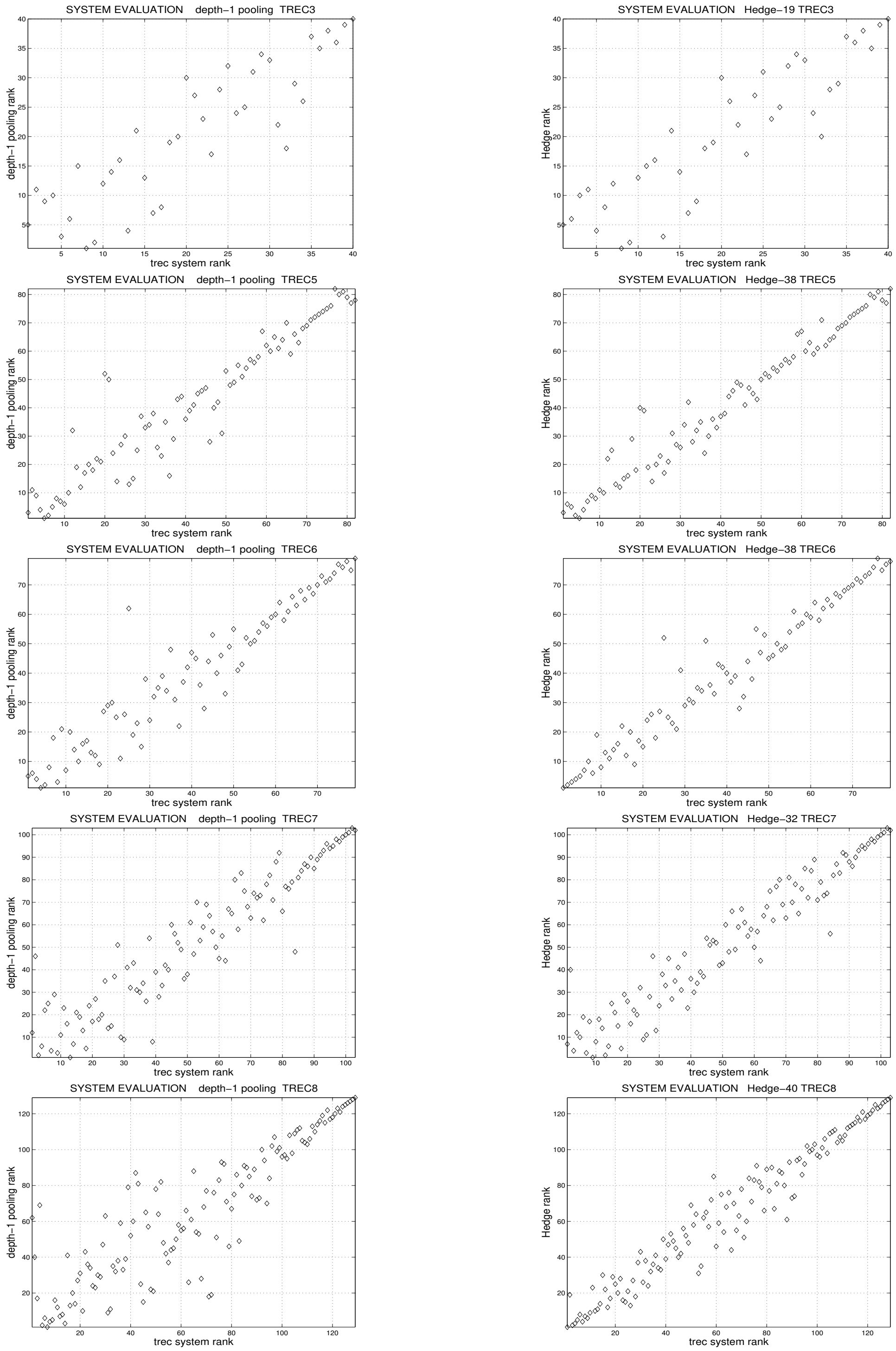

Figure 3: Depth-1 and equivalent Hedge-m rankings vs. actual ranks.

and the metasearch list in Round j will consist of these j judged documents (whether relevant or not) in order followed by the remaining unlabeled documents as ranked by Hedge. Note that the metasearch list corresponds to the user's *interactive experience*: it consists of the documents the user has judged, in order, followed by those documents the user would see if he were to cease providing feedback.

The qualities of the pool and metasearch list in any round are evaluated as follows. The metasearch list is evaluated using standard TREC routines and "ground truth" TREC relevance judgments (from the appropriate qrel file) to obtain average precision scores. The pools are evaluated in two ways. First, the fraction of total relevant documents for this query present in the pool is assessed (recall percentage). This is intended to measure the rate at which relevant documents are discovered by the evolving pools. Second, the pool is used to evaluate the underlying systems in the usual TREC manner: documents within the pool are assigned their appropriate relevance judgments, documents not in the pool are assumed non-relevant, and this "complete" set of judgments is then used to evaluate the systems to obtain average precision scores. Finally, these results are averaged over all 50 TREC queries to obtain mean average precision scores for the metasearch lists and average recall percentage scores for the evolving pools. Finally, the mean average precisions of the underlying systems induced by the evolving pools are used to rank the systems, and these rankings are compared to the "ground truth" TREC rankings of the underlying systems via Kendall's τ.

The Hedge algorithm demonstrated uniformly excellent performance across all TRECs tested (TRECs 3, 5, 6, 7, and 8) in all three measures of performance—as an online metasearch engine, as a pooling strategy for finding large fractions of relevant documents, and as a mechanism for rapidly evaluating the relative performance of retrieval systems.

In what follows, we compare the performance of the evolving metasearch list to the benchmark techniques CombMNZ and Condorcet as well as to the performance of the best underlying system in any given TREC. We further compare the performance of standard TREC-style pools to Hedge pools of an equivalent total size; i.e., if a TREC-style depth k pool contains m total judgments, it is compared to a Hedge pool with m total judgments. These pools are denoted Depth-k and Hedge-m, respectively.[5]

As shown in Table 2, the Hedge algorithm begins (in the absence of feedback) with a baseline MAP score which is equivalent or slightly better, in almost all instances, to the performance of the CombMNZ and Condorcet metasearch methods (the lower dashed lines in Figure 2(a)). As judgments are made and feedback given, the Hedge on-line metasearch results quickly surpass those of the best underlying retrieval system (the upper dashed lines). In TRECs 3, 5, and 7, the performance of the best system is equalled in 10 or fewer judgments. TRECs 6 and 8 require somewhat more judgments to achieve the performance of the best underlying system. This reflects the fact that in both cases the best systems are outliers, both in their total performance and in the documents they retrieve. Hedge must evaluate more documents to "discover" them.

Figure 2(b) demonstrates the algorithm's success in finding relevant documents. The vertical axis corresponds to recall percentage, and the dashed line indicates the performance of Depth-k pools for depths 1–10, 15, and 20 as a function of the number of total judged documents. Hedge performance far surpasses the recall rates of the depth pooling method when compared at equivalent numbers of total judged documents. But even more indicative of the success of the algorithm is a comparison of the number of judgments required to achieve equivalent recall percentages. For example, examining the TREC 8 curves along the horizontal axis, we see that the Depth-k method requires approximately 104 judgments to match the Hedge-40 return rate, and the Hedge-69 rate (36 percent) is unmatched until Depth-8 (199 judgments). After almost 500 judgments, Depth-20 has found only approximately 55% of relevant documents—a rate achieved by Hedge in less than 150 judgments.

Figure 2(c) compares the the quality of the system rankings produced by Hedge pools against those of Depth-k pools at equivalent numbers of total judged documents using the Kendall's τ measure, where ground truth is the system ordering established by TREC. Again, the dashed line indicates the results of system evaluations performed using standard TREC routines, given Depth-k pools of size 1–10, 15, and 20. Examination of TREC 8 demonstrates typical performance. At 40 documents, the τ for Hedge is 0.87. This compares with 0.73 for the Depth-1 equivalent—a substantial improvement. Likewise, Hedge-69 achieves an accuracy of 0.91 vs. a Depth-2 equivalent accuracy of 0.73.

Next, comparing along the horizontal axis the pool depths required to achieve equivalent rates of ordering accuracy, we see that to achieve an accuracy of 0.87 (Hedge-40), the equivalent Depth-3 pool requires 95 judgments. An accuracy of 0.91 (Hedge-69) is not achieved in the depth-pooling method until approximately 198 judgments (Depth-8).

Finally, a look at the scatter plots in Figure 3 demonstrates another aspect of the algorithm's performance in ranking systems—one which is somewhat obscured by the traditional Kendall's τ measure. Each pair of plots shows Depth-1 and equivalent Hedge-m predicted ranks vs. actual TREC rankings. Note in these plots that the rankings proceed from best systems in the lower left corner to worst in the upper right. TREC 3 plots are somewhat anomalous due to the relatively low number of systems in the conference, but later TRECs demonstrate the common tendency toward a difficulty in establishing proper rankings for the best systems using few relevance judgments.

While poor systems tend to be easily identified due to their lack of commonality with any other systems, the better systems tend to exhibit a similar divergence from the fold. Thus, while the rankings of poorer systems may be established using standard techniques with depth pools as small as Depth-1, the better systems (and for many purposes, the systems of most interest) tend to be the more difficult to rank correctly. As the Kendall's τ measure of accuracy in object ordering treats objects at all rank levels equally, much of the qualitative superiority of algorithms which perform well in classifying the best systems is obscured by a common tendency of most techniques to perform well on the poorer systems. Examination of tightened patterns of the Hedge plots in the region of the best systems suggests that performance of the algorithm in evaluating system orderings is somewhat better than the excellent performance demonstrated in Figure 2(c).

4. CONCLUSIONS

We have shown that the Hedge algorithm for on-line learning can be adapted to simultaneously solve the problems of metasearch, pooling, and system evaluation, in a manner which is both efficient and highly effective. In the absence of relevance judgments, Hedge produces metasearch lists whose quality equals or exceeds that of benchmark techniques such as CombMNZ and Condorcet. While in the

[5]Note that the size of a depth k pool may vary on a query-by-query (and TREC-by-TREC) basis. In any given TREC, the total size of a depth k pool over all 50 queries is calculated, and for simplicity this pool is compared to a Hedge pool containing an equal number of total judgments, spread uniformly over all 50 queries.

presence of relevance judgments, the performance of Hedge increases rapidly and dramatically.

When applied to the problems of pooling and system evaluation, Hedge identifies relevant documents very quickly, and these documents form an excellent and efficient pool for evaluating the quality of retrieval systems.

We note two possible extensions to this work. First, in all our experiments the tunable parameter β was set to 0.1. We were guided by results of Freund and Schapire in choosing this value, but we have not attempted to optimize it for our purposes, nor have we investigated its affect on our results. Our results may very well improve with a better choice of β. Second, the pools need not be generated one document at a time. At any round of the algorithm, one could output any *prefix* of the metasearch list as candidate documents for inclusion in the pool. We suspect that performance will degrade as larger prefixes are added to the pool with batch feedback, and this remains work to be explored.

5. REFERENCES

[1] *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New Orleans, Louisiana, USA, Sept. 2001. ACM Press, New York.

[2] J. A. Aslam and M. Montague. Models for metasearch. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* [1], pages 276–284.

[3] B. T. Bartell, G. W. Cottrell, and R. K. Belew. Automatic combination of multiple ranked retrieval systems. In W. B. Croft and C. van Rijsbergen, editors, *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 173–181, Dublin, Ireland, July 1994. Springer-Verlag, London.

[4] G. V. Cormack, C. R. Palmer, and C. L. A. Clarke. Efficient construction of large test collections. In Croft et al. [5], pages 282–289.

[5] W. B. Croft, A. Moffat, C. J. van Rijsbergen, R. Wilkinson, and J. Zobel, editors. *Proceedings of the 21th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, Australia, Aug. 1998. ACM Press, New York.

[6] E. A. Fox and J. A. Shaw. Combination of multiple searches. In D. Harman, editor, *The Second Text REtrieval Conference (TREC-2)*, pages 243–249, Gaithersburg, MD, USA, Mar. 1994. U.S. Government Printing Office, Washington D.C.

[7] Y. Freund and R. 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, Aug. 1997.

[8] D. Harman. Overview of the third text REtreival conference (TREC-3). In D. Harman, editor, *Overview of the Third Text REtrieval Conference (TREC-3)*, pages 1–19, Gaithersburg, MD, USA, Apr. 1995. U.S. Government Printing Office, Washington D.C.

[9] J. H. Lee. Combining multiple evidence from different properties of weighting schemes. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 180–188, 1995.

[10] J. H. Lee. Analyses of multiple evidence combination. In N. J. Belkin, A. D. Narasimhalu, and P. Willett, editors, *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 267–275, Philadelphia, Pennsylvania, USA, July 1997. ACM Press, New York.

[11] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[12] R. Manmatha, T. Rath, and F. Feng. Modeling score distributions for combining the outputs of search engines. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* [1], pages 267–275.

[13] M. Montague and J. A. Aslam. Condorcet fusion for improved retrieval. In K. Kalpakis, N. Goharian, and D. Grossman, editors, *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, pages 538–548. ACM Press, November 2002.

[14] C. C. Vogt. How much more is better? Characterizing the effects of adding more IR systems to a combination. In *Content-Based Multimedia Information Access (RIAO)*, pages 457–475, Paris, France, Apr. 2000.

[15] E. Voorhees and D. Harman. Overview of the Eighth Text REtrieval Conference (TREC-8). In D. Harman, editor, *The Eighth Text REtrieval Conference (TREC-8)*, Gaithersburg, MD, USA, 2000. U.S. Government Printing Office, Washington D.C.

[16] J. Zobel. How reliable are the results of large-scale retrieval experiments? In Croft et al. [5], pages 307–314.