Cost-Effective Inclusion of Rankers: Learning When Not To Query

Niranjan Balasubramanian, Michael Bendersky and James Allan
University of Massachusetts Amherst
140 Governors Drive, Amherst, MA 01003
{niranjan,bemike,allan}@cs.umass.edu

ABSTRACT

Combining multiple rankers has potential for improving the retrieval performance over using any of the single rankers. However, querying multiple rankers for every search request can often be expensive due to efficiency or commercial reasons. In this work, we propose a more cost-effective approach that uses the top retrieved results of a baseline ranker to predict the utility of including any additional rankers, without actually querying them. This approach is based on a novel learning technique, LTI, that maximizes the relevance gains that can be obtained by querying any additional rankers (effectiveness gains), while also minimizing the number of times these rankers are queried (efficiency gains). To evaluate LTI, we develop a combined measure that captures this trade-off in effectiveness versus efficiency. Our experimental results on a standard web collection demonstrate the viability of our approach to cost-effective ranker inclusion. Using easy-to-compute features based on the top retrieved results of a baseline ranker, the LTI method successfully adapts to different efficiency and effectiveness trade-offs and achieves on average a 7% improvement over a competitive classification baseline. In addition, we develop a method that automatically augments the training data using surrogate relevance data, which results in further effectiveness and efficiency improvements of up to 15%.

1. INTRODUCTION

Applications that combine results from multiple rankers are ubiquitous on the web today. Examples for such applications include, among others, meta-search engines (Dogpile.com, Clusty.com, Metacrawler.com), search engine comparisons (Bing-vs-Google.com) and specialized applications that use general-purpose search engines to augment their own results (Powerset.com, Facebook.com). The typical approach to leveraging multiple rankers has been to query all the available rankers. However, querying all potentially available rankers for every incoming user request can be expensive due to commercial licensing restrictions [19] or due to computational issues, in cases where rankers have high latencies.

In this paper, we propose a cost-effective ranker inclusion model for applications that leverage multiple rankers. The main idea behind our inclusion model is to utilize the results of a baseline ranker (which is always queried) to predict whether querying an additional ranker is likely to be useful. Figure 1 presents a schematic diagram of our approach. For simplicity, we assume that a system (e.g., a meta-search engine) has access to two separate black-box rankers: a base ranker B and a candidate ranker C. First a user query q is received by the system. The query is directly issued to ranker B and its results are retrieved. Based on the retrieved results a decision is made conditioned on the utility $U(q)$ of querying ranker C being greater than some threshold $T$. If this condition is satisfied, ranker C is queried and its results are merged with the base ranker results to produce a final ranking D. Otherwise, only the results of the base ranker are returned as D.

The main motivation for our approach is that the utility of including results from multiple rankers will vary for different queries. For some queries, including results from multiple rankers can result in combination of different documents, whereas for others the results from multiple rankers might have high overlap. In web search, the overlap in results from multiple rankers (inter-ranker overlap) varies significantly across different search queries, depending on query length, intent and the type of results the query retrieves. For instance, consider the two queries in Table 1. Query (a), when issued in both Google and Bing web search engines, returns the same top-3 results. On the other hand, query (b) returns completely non-overlapping results from these search engines.

Figure 1: Schematic diagram of a cost-effective inclusion of two rankers.

<table>
<thead>
<tr>
<th>Query</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>returns same top-3 results in both engines</td>
</tr>
<tr>
<td>(b)</td>
<td>returns completely non-overlapping results from different engines</td>
</tr>
</tbody>
</table>

1For instance, a query that returns a result from Wikipedia on one search engine is likely to return the same result on another.
Table 1: Top-3 results retrieved by Bing and Google search engines in response to the queries (a) david mccullough and (b) goldmine products

<table>
<thead>
<tr>
<th>Query (a): david mccullough</th>
<th>Bing Results</th>
<th>Google Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 David McCullough - Wikipedia, the free encyclopedia</td>
<td>1 David McCullough - Wikipedia, the free encyclopedia</td>
<td>en.wikipedia.org/wiki/David_McCullough</td>
</tr>
<tr>
<td>2 David McCullough</td>
<td>2 David McCullough</td>
<td>davidmccullough.com</td>
</tr>
<tr>
<td>3 David McCullough Biography</td>
<td>3 David McCullough Biography</td>
<td><a href="http://www.neh.gov/whoweare/mccullough/biography.html">www.neh.gov/whoweare/mccullough/biography.html</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query (b): goldmine products</th>
<th>Bing Results</th>
<th>Google Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 RTGroup, Inc.: Leaders In Connecting Customers for Life</td>
<td>1 Products</td>
<td><a href="http://www.frontrange.com/ProductsSolutions/">www.frontrange.com/ProductsSolutions/</a></td>
</tr>
<tr>
<td>2 Electronic Goldmine — Electronic, Circuits, Robots, ...</td>
<td>2 CRM Software</td>
<td><a href="http://www.frontrange.com/goldmine.aspx">www.frontrange.com/goldmine.aspx</a></td>
</tr>
<tr>
<td>3 Instant Product Goldmine</td>
<td>3 Gold Mine Natural Food Co.</td>
<td><a href="http://www.goldminenaturalfoods.com/">www.goldminenaturalfoods.com/</a></td>
</tr>
</tbody>
</table>

To further investigate the overlap of web search engine results for different queries, we conducted a set of experiments on a large collection of 30,000 web search queries. Our experiments showed that while the median inter-ranker overlap is below 30%, for more than 20% of the queries the overlap is higher than 60%. Such high inter-ranker overlap indicates that there will be little utility in querying an additional candidate ranker for these queries, and in such cases, querying the base ranker alone can result in substantial savings in querying costs. It is important to note that these querying costs are not limited to query response times, which can be potentially reduced by querying the candidate ranker in parallel with the base ranker, instead of in sequential order described in Figure 1. The costs of querying a candidate ranker may also include: the financial costs (for instance, a cost of querying a ranker that charges the accessing application for each individual access); the bandwidth and the network costs of communicating with the candidate ranker; and the cost of exceeding an allotted access quota (for rankers that pose restrictions on the number of accesses allowed per unit time). Since there are a large number of scenarios that can define the cost of accessing a candidate ranker, we address the most general case, and assume a fixed unit cost per access to the candidate ranker.

Motivated by the challenge of reducing the cost of accessing the candidate ranker(s), we develop a statistical model for cost-effective inclusion of rankers in a meta-search setting. In particular, we develop a learning to include model (LTI) that uses the results of a baseline ranker to predict whether the results of an additional ranker are likely to contain new relevant documents. Based on the model’s prediction of the relevance gain (the number of new relevant documents), the meta-search engine can then decide whether to query the additional ranker and include its results. While most previous work on meta-search and rank-fusion assumes that all the candidate rankers are queried all the time, using LTI, we can account for the tradeoff between the cost and the utility of querying a candidate ranker, and only choose to query it if its expected utility exceeds a defined threshold.

We develop a combined measure that captures this tradeoff between the cost and utility of querying. Our learning technique, LTI, directly optimizes for this combined measure by obtaining the best (weighted) balance between the efficiency (number of times a candidate ranker was not queried) and the effectiveness (number of additional new/relevant documents retrieved by the candidate ranker). To perform this optimization, LTI uses a set of easy-to-compute features that rely solely on the query and the output of the base ranker, with no assumptions about the inner workings of their retrieval algorithms or access to their indexes. This black-box approach is motivated by common scenarios in web search, where rankers are stand-alone search engines that expose their results through a limited-access API.

Our approach relies on training data based on manual relevance judgments, which can be expensive to obtain. To further improve LTI’s performance in scenarios with small amounts of relevance based data, we propose an automatic approach for generating surrogate training data using overlap information. This approach allows us to learn a mapping between the inter-ranker overlap and the relevance gain values. This mapping is then used to create large amounts of surrogate training data, using which we can construct a more effective (in terms of true relevance gains) model of ranker combination.

Our experiments on a standard web collection demonstrate the viability of our method for cost-effective ranker inclusion. While predicting the utility for querying the candidate ranker is a challenging task, especially given a limited amount of available relevance data, we show that we are able to improve the effectiveness of ranker inclusion, while lowering the costs of querying the candidate ranker. Further, we also show that LTI is able to adapt well to different trade-offs between effectiveness and efficiency.

The remainder of the paper is organized as follows. Section 2 covers the related work. In Section 3 we develop a model for cost-effective ranker combination. In Sections 4 and 5 we report the experimental setup and the experimental results. We draw the conclusions in Section 6.
2. BACKGROUND AND RELATED WORK

Combining results from multiple search engines has been studied in diverse application settings such as meta-search [22], rank-fusion [2, 14] and distributed search [8]. A complete survey of this work is beyond the scope of our paper. Instead, we focus on approaches that pertain to cost-effective combination of search results in a black box meta-search scenario that is common on the web today. In particular, we present prior work that discusses the effectiveness of ranker combination in relation to diversity in results of the rankers, and work that focuses on the effectiveness versus efficiency trade-offs in combining results from multiple search engines.

2.1 Effectiveness

In the meta-search setting, multiple rankers can be used to either fuse search results to produce a new ranking [22], or to route the user to the most effective search engine [26].

Rank fusion approaches combine evidence from multiple search engines in order to create a fused rank list [2, 18, 23, 13, 14]. There are two possible reasons for improvements obtained by rank fusion approaches. First, if there is high overlap of relevant documents in the search results compared to non-relevant overlap, the combination of evidence will favor relevant documents more and improve the precision of the fused results [16]. Second, the search results can contain different relevant documents, whose combination improves the recall of the fused results. Bietzel et al [5, 6], show that when the retrieval systems used are highly effective but have systemic differences3, the improvements due to fusion are largely due to improvements in recall. In the web meta-search scenario, where the rankers are both effective and diverse, rankers with low overlap in search results are more likely to produce better fused results compared to rankers with higher overlap [12].

Different search engines can provide higher quality results for different queries. White et al [26] develop query and result-set dependent techniques to solve the problem of automatically routing users to the search engine that provides the best result for a given query. We consider a variant of this problem, where suggesting the search results of an additional search engine is useful only if the additional search results contain new relevant documents. Further, we consider the practical meta-search setting which imposes further efficiency restrictions. In particular, we consider the scenario where a meta-search engine has access to the search results page alone and needs to minimize the number of queries issued to rankers.

Thus, predicting the overlap in search results can be useful for effective and efficient combination of these rankers in both rank fusion and routing scenarios. In this work, we focus on a measure that is proportional to the number of new relevant documents that can be obtained by using an additional search engine. However, the techniques and features we propose are also useful for predicting the general overlap in search results of multiple rankers.

2.2 Effectiveness versus Efficiency

Prior work on federated search has focused on the effectiveness versus efficiency trade-offs involved in selecting a small number of resources that maximize the number of relevant documents returned. Selecting too few resources might yield low recall, whereas selecting too many resources can be inefficient. Si et al [24] utilize a centralized sample of documents created from past queries to estimate the relevance of documents obtained from individual search engines. The estimated relevance of the top-k documents from each search engine is then used in an optimization framework to determine the smallest set of search engines that maximize the expected utility. Cetintas et al [9] propose an extension which considers the cost of downloading search results. Using an optimization framework that addresses the trade-off between effectiveness and efficiency their approach determines the number of documents to download in order to assess the utility of results merging. More recently, Arguello et al [1] use several corpus dependent, query-category based, and click-based features to select few sources (collections) whose results can be combined with effectiveness comparable to a full retrieval on all collections.

In a web search setting, Baeza-Yates et al [4] investigate the effectiveness versus efficiency issues in a two-tiered search model with a local server, and a remote server. Usually, every query is routed to a local server first, and is routed to the secondary server only if the results from the local server are deemed inadequate. To avoid the poor response times due to sequential querying, they propose an approach that is able to predict whether the local server’s results will be sufficient prior to retrieval.

Our work differs from these federated search approaches both in terms of the additional constraints imposed by the black box metasearch scenario and the techniques employed. First, in contrast to the standard federated search setting [1, 4, 3, 9], each query to a ranker incurs a cost regardless of the number of top-k search results obtained, and the meta-search engine has no direct access to full document texts and rankers indexes. In addition, instead of just relying solely on inter-ranker overlap, as is done in some previous work [4], we utilize a relevance gain metric when relevance judgments are available. Finally, we propose to leverage the overlap in order to automatically create surrogate training data in cases when relevance judgments are scarce.

3. LEARNING TO INCLUDE RANKERS

In this paper we focus on the standard meta-search scenario, where our (meta) search engine has access to rankers that retrieve and rank documents from different (but potentially overlapping) collections. Typically, a meta-search engine always queries all the rankers and fuses the returned results [2, 22]. However, querying all the available rankers all the time can be expensive due to licensing restrictions. For example, some search API’s already impose time-based or call-based quotas [19]. In some cases, accessing all the rankers can also be time-consuming. For instance, one or more rankers may have high latencies, thus hurting the overall response times. Instead of always accessing all the rankers, we propose a more cost-effective approach that always uses a single base ranker, and only accesses additional ranker(s) when its expected utility is above a certain threshold.

In this section, we formally define the learning to include (LTI) problem, develop an evaluation measure, and describe an approach for solving the LTI problem.

3.1 Problem Definition

We make the following simplifying assumptions in defining
the learning to include problem:

1. Our meta-search engine has access to two rankers: (i) a base ranker, $B$ and (ii) a candidate ranker, $C$. The base ranker, $B$, is always queried. The candidate ranker, $C$, is queried only if some criteria is met. Note that while conceptually simple, this setting can be easily extended. For instance, we can extend it to a case of multiple rankers by treating $C$ as a set of candidate rankers.

2. We are interested in a black-box scenario, a very common scenario for meta-search engines on the web, in which the search engine has no access to the internal workings of its rankers such as their retrieval algorithms or their indexes. Instead, the meta-search engine can only submit a query $q$ to a ranker $r$, and get in response a ranking $D_r$, containing $K$ retrieved results. Ranking $D_r$ typically includes an ordered list of links to retrieved documents (URLs in web search), each accompanied by a brief snippet that provides a short query-biased preview of the document content.

3. Instead of choosing a specific cost setting such as financial costs imposed by licensing restrictions or network costs, we assume a general cost setting, where every time we use the candidate ranker $C$, we incur a fixed (unit) cost.

In this setting, the learning to include problem is to learn a decision function $I$, which given a query $q$, and the results of the base ranker, $B$, indicates whether the candidate ranker, $C$ should be queried as well to obtain additional results for inclusion. The main goal of LTI is to choose a decision function that maximizes the gains that can be obtained by querying the candidate ranker (effectiveness gains), while also minimizing the number of times the candidate ranker is queried (efficiency gains).

### 3.2 Evaluation Measure

Since we are interested in obtaining the best possible trade-off between the competing effectiveness and efficiency gains, we evaluate the decision functions for LTI using a weighted harmonic measure that combines the two.

Formally, let $Q$ be a set of queries over which we measure the trade-off in effectiveness versus efficiency and let $I(q, D_B)$, be the indicator function which indicates whether, given a query $q$ and a base ranking $B$, we query the candidate ranker $C$. Using this notation, we first define the effectiveness, and efficiency measures, which we then use to define the combined measure.

**Effectiveness:** The effectiveness of the indicator function, $\text{effect}(I)$, is measured as the fraction of possible gains in relevance that are achieved when querying the candidate ranker in accordance to $I$. Letting $G(q)$ denote the relevance gain obtained when querying the candidate ranker for query $q$, we define the effectiveness of $I$ as follows:

$$\text{effect}(I) = \frac{\sum_{q \in Q} I(q, D_B) \times G(q)}{\sum_{q \in Q} G(q)}$$  (1)

For each query $q$, we define the gains obtained when using the candidate ranker, $G(q)$, in terms of the new relevant documents retrieved by the candidate ranker $C$. Let $R_B$ and $R_C$ denote the sets of relevant documents in the top $K$ results retrieved by $D_B$ and $D_C$, respectively. Then, we define relevance gain for query $q$ as:

$$G(q) = \min\left(\frac{|R_C \cap R_B|}{|R_B|}, 1 - \frac{|R_B| - |R_C|}{|R_B|}\right)$$  (2)

Thus, $G(q)$ measures the number of additional relevant documents that can be added to the top $K$ ranks of the baseline ranking, if we had an optimal combination algorithm. If the baseline ranking is already optimal (i.e., all top $K$ ranks in the baseline were relevant), then querying the candidate ranker $C$ cannot yield any additional improvements to the top $K$ ranks. On the other hand, if the baseline ranking is not optimal, then $C$ can contribute new relevant documents through combination.

Note that $G(q)$ represents the optimal bound (in terms of prec@K) on the relevance gain from combining the rankings $D_B$ and $D_C$. Actual combination of $D_B$ and $D_C$ using existing techniques such as CombMNZ [14], Borda-fuse [2] or probFuse [18] may not achieve this bound, but – as previous work indicates [6] – their performance is likely to correlate with it.

**Efficiency:** According to our definition of the LTI problem, every time the candidate ranker $C$ is queried, we incur a fixed unit cost (as per the third assumption in Section 3.1). This suggests a straightforward definition of efficiency gains as the proportion of times we avoid querying the candidate ranker when using $I$. Formally, $\text{effic}(I)$ is defined as follows:

$$\text{effic}(I) = 1 - \frac{\sum_{q \in Q} I(q, D_B)}{|Q|}$$  (3)

**Combined Measure:** Using the effectiveness and efficiency definitions in Equations 1 and 3, we define the combined measure for evaluating the indicator function $I$ as the harmonic mean of the two:

$$\mathcal{E}_\alpha(I) = \frac{\text{effect}(I) \times \text{effic}(I)}{\alpha \times \text{effect}(I) + (1 - \alpha) \times \text{effic}(I)}$$  (4)

We use the weighted harmonic mean, as it amplifies the contributions of the outliers (extremes) in both effectiveness and efficiency. For example, when $\alpha$ is set to 0.5, $\mathcal{E}_{\alpha=0.5}$ does not favor solutions that are highly inefficient even when they are highly effective and vice versa. To achieve a high $\mathcal{E}_{\alpha=0.5}$ a solution must achieve a good effectiveness and efficiency balance.

### 3.3 Threshold-Based Classification

Our goal is to optimize the indicator function $I(q, D_B)$ such that the combined measure, $\mathcal{E}_\alpha$, (as defined in Equation 4) is maximized. Note that in Equation 4, $\alpha$ is a free parameter that can be used to control the relative importance of effectiveness and efficiency. Higher $\alpha$ values favor efficient solutions, whereas lower $\alpha$ values favor effective solutions. $\alpha$ can be chosen depending on the constraints imposed on the meta-search engine such as the cost of querying the candidate rankers or the amount of queries that can be issued to the candidate rankers in a given time period. In this paper, we focus on developing a general solution that can maximize this combined measure $\mathcal{E}_\alpha$, for any given $\alpha$.

To this end, we propose a threshold-based classifier. Intuitively, the candidate ranker $C$ should be included when it is
likely to be useful, and avoided otherwise. The usefulness of querying the candidate ranker depends on the setting of $\alpha$, which controls the relative importance of effectiveness and efficiency goals. Given some predictions about the likely gains for each query, for large values of $\alpha$, when efficient solutions are preferred, querying $C$ is useful only if the predicted probability of the effectiveness gain is large. On the other hand, for small values of $\alpha$, which prefer effective solutions, querying $C$ is useful even for small (but non-zero) predicted probabilities of effectiveness gain.

Based on this intuition, we build a binary classifier that predicts for each query, whether the gains from using the candidate ranker will exceed a specified threshold $T$. Instead of directly using the classifier’s decision, we use the classifier’s output probability, $P(G(q) \geq T)$, which represents the probability of the gain measure, $G(q)$ being above a given threshold $T$ to build a threshold-based indicator function, $I$ as follows:

$$I(q, D_B) = \begin{cases} 1 & \text{if } P(G(q) > T) \geq p \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

If the output probability of the classifier is greater than $p$, then we query the candidate ranker $C$. Otherwise, we use the base ranker alone. Clearly, the performance of the indicator function $I$ will vary for different choices of $p$. Higher $p$ values reduce the number of times $C$ is queried (i.e., favoring efficient solutions), whereas lower $p$ values increase the number of times the $C$ is queried. Accordingly, for a given $\alpha$, $p$ values that maximize $\mathcal{E}_\alpha$ can be learned.

Clearly, the performance of the binary classifier, which determines the probability $P(G(q) \geq T)$, will influence the combined measure $\mathcal{E}_\alpha$ that can be achieved by applying the indicator function $I$. From Equation 5 we can see that low classification errors will correspond to high $\mathcal{E}_\alpha$ values. Accordingly, to optimize the performance of the binary classifier we set the threshold $T$ in Equation 5 to be the median of the gains on a training set of queries. This setting ensures that the number of positive ($G(q) > T$) and negative ($G(q) \leq T$) examples in the training set is balanced, a desired property from a classification standpoint.

### 3.4 Features

To learn a model that predicts the probability $P(G(q) > T)$ (Eq. 5), we train a standard logistic regression model

$$P(G(q) > T) = \frac{1}{1 + e^{-\Lambda}},$$

where $F_q = f_{q1}, \ldots, f_{qn}$ is a feature vector representing query $q$ and $\Lambda = \lambda_1, \ldots, \lambda_n$ is an associated weight vector, which is optimized to reduce the classification error on a training set.

Table 2 shows the features in $F_q$. We divide the features into two groups based on their source—the query itself, and the retrieved set $D_B$. We can either use only the features based on the query itself, and perform a utility prediction without querying the base ranker, or allow for querying the base ranker and use features from both sources.

There are two main motivations for each feature we use: (a) it has to be correlated with the expected utility of querying a candidate ranker $C$ for query $q$, and (b) it has to be highly efficient to compute. For some features that are computed over the retrieved list $D_B$, we compute several aggregates (see Table 2). Each of these aggregates is used as a separate feature in $F_q$.

In contrast to previous work we use neither the traditional pre-retrieval query performance predictors such as IDF or PMI [15, 4] nor post-retrieval performance predictors such as Query Clarity [11] or Weighted Information Gain [27]. This is due to the fact that we restrict our attention to the black-box setting assuming that we have no access to rankers’ retrieval algorithms or indexes.

Instead, we use only the information we can glean from the query itself, such as its length and its grammatical structure (e.g., features $\text{qLenTerms}, \text{qLenChars}, \text{qIsCap}, \text{qIsQuestion}$), which were shown to correlate with query performance [7], the structure of URLs (features $\text{uDepth}, \text{uLenChars}$) and the contents of the snippets in the retrieved list. To estimate inter-ranker overlap, we use the intra-ranker overlap (overlap between the retrieved snippets - $\text{uuOverlap}, \text{uuEntropy}$) and query-ranker overlap ($\text{uNgramCover}, \text{uFullCover}$) as approximations.

### 3.5 Learning with Surrogate Gains

Thus far we have only considered including rankers using the definition of relevance gain in Equation 2, which requires training data in form of documents manually judged for relevance for a particular query $q$. However, in practice, the number of judged relevant documents for learning a ranker inclusion model based on the relevance gain $G(q)$ is limited.

To overcome this problem, we propose the use of overlap between the two ranked lists—the number of documents in common, as a surrogate for relevance gain. Given a query $q$ and the two rankings $D_B, D_C$ we define overlap $O(q)$, as a fraction of the results in $D_B$, which appear both in $D_B$ and in $D_C$.

$$O(q) = \frac{|D_B \cap D_C|}{K} \quad (6)$$

Since computing the overlap does not require relevance judgments, we can automatically generate large amounts of this surrogate data.

However, directly utilizing the overlap information in place of the relevance based gain is not viable because low overlap between retrieved sets does not always correspond to high relevance gains from their combination [17]. To illustrate this point, Figure 2 shows the distribution of relevance gains against different levels of overlap between two different retrieval systems for 150 Gov2 TREC topics. As expected, when overlap is high the possible gains are usually low. However, when the overlap is low there is a much higher variance in possible gains.

Instead of directly using overlap in place of relevance gain, we first learn a model that learns to map the overlap information and the relevance gains using the data for which relevance judgments are available. Then, for each instance in the automatically generated surrogate data, we obtain the relevance gains predicted using this learned model. This surrogate data now augmented with predicted relevance gains is then combined with the original data for training. We formally define this process of learning with surrogate gain data as follows.

Let $G$ denote the set of training queries that have both relevance gain ($G(q)$) information and overlap ($O(q)$) information. In addition, let $O$ denote the set of training queries that only have overlap information but no relevance gain information. Let $F_q$ denote the features associated with each
query \( q \) (as described in Section 3.4). To obtain the surrogate gains, we first learn a mapping function

\[
M : \{ F_q, O(q) \} \rightarrow G(q),
\]

using linear regression on \( G \) that is used as the training data.

Then, we create an augmented training set \( G' \), by applying the mapping function to \( O \) i.e.,

\[
G' = G \bigcup_{q \in O} M(F_q, O(q)). \tag{7}
\]

Finally, we learn the ranker inclusion model using this augmented training set. It is important to note that in this process of learning with surrogate gain data, the ranker inclusion model in Equation 5 is still trained over the same set of features as before and does not use overlap as a feature, since it is not available during testing.

4. EXPERIMENTAL SETUP

4.1 Datasets and Rankers

We evaluate our approach using two datasets (1) Million and (2) Gov2. Million is a set of 10,000 title portions of topics created for the TREC 2008 Million Query Track and Gov2 is set of 150 title portions of topics created for the TREC Terabyte track. Both Million and Gov2 queries were used over the same document collection (a crawl of the .gov domain). In keeping with the black-box scenario for meta-search, we retrieve the top 10 search results for each query and only extract the corresponding URL, and the snippet information found on the results page. The queries in the Gov2 dataset are used for all the experiments, and the queries from the Million dataset are used for the experiments in learning with surrogate gains.

For the retrieval experiments, we use Indri [25], and use Query Likelihood (QL) [20] as a baseline ranker, and Okapi BM25 [21] as a candidate ranker. We choose Okapi BM25 as the candidate ranker as it has a higher retrieval effectiveness (in terms of mean average precision) over QL, and provides significant relevance gains, \( G(q) \), for a larger number of queries. We use the default snippet generation available in Indri.

In all the experiments, we report the results obtained using a 3-fold cross-validation.

4.2 Tasks

We conduct two sets of experiments to validate our approach for predicting the utility of the inclusion of the candidate ranker. The experiments are set up as follows.

1. Gain-based ranker inclusion. The objective for this task is to select queries for which the candidate ranker \( C \) is to be queried using the threshold-based classification approach described in Section 3.3. We consider only the top 10 search results from each ranker and compute the gain measure, \( G(q) \) with \( K = 10 \). For this task, we only use the 150 queries from the Gov2 collection, for which relevance judgments are available.

2. Ranker inclusion using surrogate data. Similar to the first task, the objective for this task is to select queries for which the candidate ranker \( C \) is to be queried. However, in addition to the relevance gain data (Equation 2) for the 150 queries from the Gov2 collection, we also use the overlap data (Equation 6) from the 10,000 queries in the Million collection. The ranker inclusion model is learned using the combination of both true relevance gain data and the surrogate gains, which are learned according to the mapping \( M \) described in Section 3.5. Using this mapping, we map the overlap values, \( O(q) \), for Million queries (\( O \)) to predicted gain values, \( M(q) \), and use them to gener-
5. EVALUATION

5.1 Gain-Based Ranker Inclusion

5.1.1 Optimizing $\epsilon_\alpha$

We evaluate the threshold-based approach for the task of deciding when to query the candidate ranker. As a baseline, we use a standard binary classification solution, Classifier, which is trained using the same set of features as the threshold-based classification approach. Similar to the threshold-based approach, Classifier first predicts the probability that the gain for a given query exceeds the specified threshold $T$. However, unlike the thresholded approach, Classifier uses a fixed cutoff $p = 0.5$, on the predicted probability to decide whether to query the candidate ranker (see Equation 5). For a given query, Classifier includes the candidate ranker only if the output probability for the query exceeds this fixed cutoff 0.5. However, for different values of $\alpha$ in $\epsilon_\alpha$, this fixed setting of $p$ can either be too conservative (if highly effective solutions are preferred) or too lax (if highly efficient solutions are preferred).

In contrast, our threshold-based approach (denoted LTI) can learn different values of $p$ for different settings of $\alpha$. Accordingly, for a given value of $\alpha$ we pick the value of $p$ from the range $[0, 0.05, \ldots, 0.95, 1]$ which yields the highest $\epsilon_\alpha$ on the training queries.

Figure 3: Comparison of the dynamic $p$ setting (LTI) and the fixed binary classifier for different $\alpha$ values.

Figure 3 shows that for most values of $\alpha$, LTI outperforms the baseline, Classifier. For most values of $\alpha$, the setting of $p = 0.5$ used by the Classifier is sub-optimal, in terms of $\epsilon_\alpha$, since it assumes a fixed relative importance between effectiveness and efficiency. This is especially evident for the cases where $\alpha$ is either very large ($\alpha \geq 0.9$) or very small ($\alpha \leq 0.2$) — that is in cases where effectiveness and efficiency are not equally important.

Table 3 compares the performance of the two methods in terms of effectiveness and efficiency for different values of $\alpha$. When $\alpha$ is 0.1, LTI increases the number of times the candidate ranker is queried, and thus has higher effectiveness (at the cost of lower efficiency). On the other hand, when $\alpha$ is set to 0.9, LTI reduces the number of times the candidate ranker is queried and thus, increases efficiency (at the cost of lower effectiveness). In contrast, the the fixed binary classifier always achieves the same effectiveness-efficiency tradeoff (since $p$ is fixed at 0.5), and does not adapt well to different $\alpha$ settings.

Figure 4: LTI performance for varying $\alpha$.

The plot in Figure 4 further illustrates how LTI learns different effectiveness-efficiency tradeoff strategies. For low $\alpha$ values LTI favors effectiveness and as $\alpha$ value increases, LTI starts to favor efficiency more at the cost of lower effectiveness. Also, LTI directly optimizes for $\epsilon_\alpha$, the harmonic mean of the effectiveness and efficiency measures, which discourages extreme solutions. As a result, we find that for nearly all settings (except for $\alpha = 1$) LTI does not favor extreme solutions, where either efficiency or effectiveness is too low.

Figure 5: Effectiveness for different efficiency requirements: Plot that shows percentage improvements in effectiveness over a random choice of queries at different efficiency requirements.

5.1.2 Threshold-Based Performance Analysis

To better understand the performance of LTI under different performance regimes, we plot the effectiveness gains that are possible under a given efficiency requirement. An efficiency requirement is defined as the percentage of times the candidate ranker is queried. To compute the effectiveness gain possible for a given efficiency requirement, we sort the queries based on the effectiveness gain for these queries predicted by the LTI$^3$. Then, for a given efficiency require-

$^3$Note that LTI only predicts probabilities of gains exceeding
Table 3: Comparison of the fixed binary classifier and the dynamic $p$ setting (LTI). Combined measure $\mathcal{E}_\alpha$, effectiveness and efficiency are reported for different values of $\alpha$.

<table>
<thead>
<tr>
<th>Method</th>
<th>$\alpha=0.1$</th>
<th></th>
<th>$\alpha=0.5$</th>
<th></th>
<th>$\alpha=0.9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classifier</td>
<td>0.3908</td>
<td>0.3744</td>
<td>0.6443</td>
<td>0.3744</td>
<td>0.6443</td>
</tr>
<tr>
<td>LTI</td>
<td>0.5642 (+44%)</td>
<td>0.5806</td>
<td>0.4497</td>
<td>0.5079 (+7.2%)</td>
<td>0.4668</td>
</tr>
</tbody>
</table>

We plot the difference between the LTI gains and the random gains as the percentage of additional gains for different efficiency requirements as shown in Figure 5. For most efficiency requirements, LTI provides substantial additional gains compared to a random choice, except at the extremes. At one extreme, when only a small percentage of the queries (less than 10 queries) can be sent to the candidate ranker, the resulting gains are not significant in both the random case and the LTI case. At the other extreme, when nearly all the queries can be routed to the candidate ranker, any random choice of queries is likely to be close to the best possible choice of queries. However, in the wide range of efficiency levels between 20% and 90%, selecting queries using the LTI method yields significant effectiveness gains compared to the random choice.

5.1.3 Query Examples

It is interesting to examine the performance of the LTI method for different queries. In particular, in Table 4, we show the queries for which our method has the highest and the lowest accuracy in predicting the effectiveness gain stemming from including the results of the candidate ranker. In our query-by-query analysis, we were not able to identify a particular type of queries for which our method always has either very good or very bad prediction performance. As can be seen from Table 4, the best and the worst performing queries do not vary significantly in either their length or their grammatical structure.

One interesting observation in Table 4 is that, in general, for the best performing cases (the top half of the table), our method tends to predict correctly the cases in which no major gain is expected as a result of the inclusion of the results from the candidate ranker. These are either the queries with the best retrieval performance (for which the base ranker already has perfect performance) or the queries with the worst retrieval performance (for which both the base ranker and the candidate ranker fail to retrieve relevant documents). On the other hand, the worst performing cases (the bottom half of Table 4), are split between cases of extremely high and low relevance gain as a result of the inclusion of the candidate ranker.

Table 4: Examples of queries for which the highest and the lowest gain prediction performance is observed.

<table>
<thead>
<tr>
<th>Query</th>
<th>Predicted</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>nuclear reactor types</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>david mccullough</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>hunting deaths</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>urban suburban coyotes</td>
<td>0.09</td>
<td>0.1</td>
</tr>
<tr>
<td>low white blood cell count</td>
<td>0.19</td>
<td>0.2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>artificial intelligence</td>
<td>0.78</td>
<td>0</td>
</tr>
<tr>
<td>history of physicians in america</td>
<td>0.83</td>
<td>0</td>
</tr>
<tr>
<td>big dig pork</td>
<td>0.93</td>
<td>0.1</td>
</tr>
<tr>
<td>executive privilege</td>
<td>0</td>
<td>0.9</td>
</tr>
<tr>
<td>civil air patrol</td>
<td>0</td>
<td>0.9</td>
</tr>
</tbody>
</table>

5.1.4 Feature Analysis

In this section, we analyze the correlation between the features used to predict the effectiveness gain, and the gain itself. The features that we use to predict the effectiveness gains are described in Table 2. In Table 5, we show the ten features with the highest value of Spearman’s $\rho$ [10], which demonstrates the rank correlation between these features and the effectiveness gain expected from including the results from the candidate ranker.

Note that the most predictive features for estimating effectiveness are based on the result set of the base ranker, rather than the query itself. In fact, the three query dependent features in the Table 5 occupy the lowest positions in the table.

The rest of the features in Table 5 are based on the results retrieved by the base ranker. Different aggregates of the inter-snipplet overlap ($uuOvlp$) within the base result set are inversely proportional to the expected effectiveness gain. That is, a query that results in highly similar results in the base result set is unlikely to benefit from including the results from the candidate ranker. On the other hand, a query that returns potentially relevant, but diverse results is likely to benefit from including an additional ranker. This is demonstrated by the positive correlation of the $uuOvlp$ feature, and the negative correlation of the $uuOvlp$ feature.

It is important to note, however, that in all the cases, the values of Spearman’s $\rho$ are relatively low, indicating that each feature on its own is not a very reliable predictor of the expected gain. This indicates that our task of learning a cost-effective combination of the base and the candidate
rankers is challenging, especially given the limited amount of training relevance data. Therefore, in the next section we explore the benefits of using surrogate data to improve the performance of our ranker inclusion approach.

5.2 Ranker Inclusion Using Surrogate Data

5.2.1 Varying the Amount of Surrogate Data

To demonstrate the utility of using surrogate data for the gain-based ranker inclusion, we conduct experiments by creating an augmented training set $G'$, according to the procedure described in Section 3.5. To create the augmented training set we use samples of varying size from the Million collection. Figure 6 shows the performance of LTI in terms of $E_\alpha$ with $\alpha = 0.5$, when different amounts of surrogate instances are added to the training data. Adding surrogate instances provides substantial improvements in $E_{\alpha=0.5}$, ranging from 2% to 8%, over the threshold-based approach trained without any surrogate instances.

![Figure 6: Effect of adding increasing amounts of transfer instances on the $E_\alpha$ measure. The horizontal line corresponds to the performance of LTI with no transfer instances from Million.](image)

5.2.2 Detailed Performance Analysis

Adding surrogate instances also provides improvements in $E_\alpha$ for most settings of $\alpha$. Figure 7 shows the performance of the threshold-based approach when training with all 10000 surrogate instances. For lower $\alpha$ values, $0.1 \leq \alpha \leq 0.6$, adding surrogate instances provides improvements ranging from 8% to 15%. However, for higher values of $\alpha$, $\alpha \geq 0.6$, adding surrogate instances does not provide substantial improvements.

Overall, adding surrogate instances increases the amount of training data for both 1) the classifier that predicts the probability of the gains exceeding a given threshold, and 2) for learning the cutoff $p$, on the predicted probabilities. As a result, we find that the classification accuracy – the accuracy of predicting whether the gains exceed a specified threshold – improves substantially, which in turn improves the performance with respect to $E_\alpha$.

A closer inspection shows that the improvements in accuracy are mostly due to improvements in recall – that is adding surrogate data causes more queries whose actual gains exceed the specified threshold to be identified. However, because the surrogate instances are noisy training data, their addition also leads to false positives – that is it causes some decline in precision. As a result, adding surrogate instances mainly tends to improve effectiveness, while slightly lowering efficiency.

Since at lower $\alpha$ settings, effective solutions are preferred, the addition of surrogate data leads to substantial improvements in $E_\alpha$. For instance, at $\alpha = 0.1$, we see nearly a 15% improvement. On the other hand, for the higher $\alpha$ settings, which favor efficiency, we do not observe such large improvements in $E_\alpha$.

![Figure 7: Performance of LTI with and without surrogate instances for different $\alpha$ settings.](image)

6. CONCLUSIONS AND FUTURE WORK

In this paper, we discuss a problem of ranker combination in a black-box setting, where the meta-search engine has no access to the internal information about its rankers. To measure the performance of ranker combination, we propose a measure $E_\alpha$ that balances the trade-off between the effectiveness and the efficiency aspects. We develop LTI, a statistical model for cost-effective ranker combination that directly optimizes the proposed measure.

Empirical results on a standard web collection demonstrate the utility of our approach. Compared to the standard classification method, LTI provides notable improvements in both the effectiveness and the efficiency of the resulting ranker combination. These improvements are consis-
tent for different effectiveness and efficiency balances. For the cases when the available relevance data is scarce, we develop a technique for automatically generating surrogate training data using the ranker overlap information. Our experimental results show that surrogate data can improve the performance of the LTI method by as much as 15%.

In this work, we operated within the constraints imposed by the black box scenario, with no access to large external collections or search history. In this scenario, we must query the base ranker in order to obtain the reliable features for learning the ranker combination. A natural extension to this work would be to partly relax the black-box constraints and to allow the meta-search engine to use external data when available, in order to avoid always querying the base ranker.

7. ACKNOWLEDGMENTS

This work was supported in part by the Center for Intelligent Information Retrieval and in part by ARRA NSF IIS-901442. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

8. REFERENCES