Learning to Merge Search Results for Efficient Distributed Information Retrieval

Kien-Tsoi T. E. Tjin-Kam-Jet
University of Twente, The Netherlands

Djoerd Hiemstra
University of Twente, The Netherlands

ABSTRACT
Merging search results from different servers is a major problem in Distributed Information Retrieval. We used Regression-SVM and Ranking-SVM which would learn a function that merges results based on information that is readily available: i.e. the ranks, titles, summaries and URLs contained in the results pages. By not downloading additional information, such as the full document, we decrease bandwidth usage. CORI and Round Robin merging were used as our baselines; surprisingly, our results show that the SVM-methods do not improve over those baselines.

Keywords
Distributed information retrieval, results merging, interleaving, round robin, learning to rank, meta-search, federated-search, collection fusion.

1. INTRODUCTION
Centralized search is limited by its inability to search through the *deep web*—pages accessible only after querying an HTML form, since web crawlers lack the intelligence to adequately fill in and submit such forms. Another drawback is that the index needs to be maintained and updated to cope with both content change and Web growth [1].

With Deep Web content already residing in searchable databases, and in the expectation that the Web will continue its enormous growth, a promising search paradigm is DIR (Distributed Information Retrieval) [3]. A DIR system contains at least one broker and multiple servers, each indexing its own document collection. The broker serves as a mediator between the user and the servers. The user sends a query to the broker, which subsequently selects the servers most capable of adequately answering the query, and forwards the query to the selected servers. Each server then retrieves its most relevant documents and sends a ranked list of results back to the broker, which merges these into one results list and presents it to the user. Generally, although the broker only controls the way in which the servers are selected and the way their results are merged, it has no control over the internal functioning of any server.

DIR is a well-established research area with three main areas of interest: server description, server selection, and results merging [3]. A server description is often an excerpt of a server's index and it is used to estimate the number of different words and word frequencies of the server [4, 11]. In this way, server selection is done by treating each excerpt as one very large document, and subsequently applying standard IR technology to rank and select the top N servers. Most existing result-merging methods require the server to supply a document score—otherwise an estimate of the score is used. These scores are then adapted so that inter-server document scores can be compared and ranked. However, in practice, document scores are hardly ever provided by search servers, or if they are, they cannot be trusted.

In this paper, we propose the use of information from search result snippets that search servers typically provide: the document title, its url, and a dynamically generated document summary containing the matching query terms. Unlike in previous work, our broker does not have any excerpt of any server's index, nor does it require document scores to be supplied along with the server's results. Therefore, we apply methods that neither rely on estimated indices and document scores, nor on the download of any additional information, such as the full document. We use SVM [16] (Support Vector Machine) to train a function for merging results based only on evidence contained in the results pages received from the servers. In addition, the benefit of not downloading any additional information is decreased bandwidth usage.

Outline of paper: Section 2 summarizes key literature about results merging. Our experiment testbed is explained in Section 3. Section 4 presents our merging approach, and the evaluation is discussed in Section 5. Section 6 presents and discusses our results, and Section 7 gives our conclusion.

2. RELATED WORK
2.1 CORI
CORI [3, 5] has been used by many researchers [7, 9, 8, 13] as a baseline for server selection and results merging. Query Based Sampling (QBS) [4] is often used to obtain the server descriptions needed to run the CORI server selection algorithm which ranks the servers based on the belief-score

of observing the query's terms in that particular server.

Once the results pages are obtained from the selected servers, the document scores given by the distinct servers are normalized and weighted as follows:

$$w = 1 + 0.4 * \frac{s - S_{min}}{S_{max} - S_{min}} , \quad (1)$$

$$D' = \frac{D - D_{min}}{D_{max} - D_{min}} , \quad (2)$$

$$D'' = \frac{D' * w}{1.4} . \quad (3)$$

where s is the server's belief score; S_{min} and S_{max} are the highest and lowest belief scores respectively that CORI could potentially assign to a server; D is the document score supplied by the server; D' is the normalized document score; and D'' is the weighted document score.

Note that (2) requires cooperation among servers because D_{max_i} and D_{min_i} must be provided by the server when it returns document rankings. Our goal is *not* to rely on any form of cooperation, because cooperation can be unreliable in multi-party environments. In the absence of cooperation, D_{max_i} is set to the maximum document score returned by the server and D_{min_i} is set to the minimum [13].

2.2 Ranking-SVM

Joachims developed an SVM-type called Ranking-SVM [6], he used it to learn a preferred ranking function from clickthrough data. He argued that clickthrough data can be recorded at very low cost, and that users make a (reasonably) informed choice when clicking on a link, instead of clicking at random. Therefore, clicks are likely to convey some partial ranking information that can be used to learn a ranking function.

For example, if a user clicked on results 3 and 5, the preferred ranking would be: 3,5,1,2,4. In other words, the system made some errors: it should have ranked result 3 ahead of results 1 and 2, and result 5 ahead of results 1, 2 and 4. These five errors, called preference constraints, are deduced from the clicks (plus the ranked list) and serve as the input for the SVM^{light} program that Joachims developed. The input consists of (labeled) document pairs, where one document is preferred over the other. The program tries to learn a ranking function that maximizes the proportion of correctly-ordered pairs of documents (induced by the learned ranking function when compared to the preferred rankings).

2.3 Regression-SVM

Several researchers [7, 10, 12] tackled the results merging problem by learning a regression function that maps server-specific document scores to centralized document scores—centralized scores are derived from a central index that contains many sampled documents from all servers. The motivation behind this approach is that the document scores produced by all servers are usually incomparable.

Inspired by this approach, we decided that, instead of mapping server-specific document scores to centralized document scores, we could use Regression-SVM to learn a function that directly determines the "centralized" score of a document, given its features.

3. TESTBEDS

We used the multi-purpose TREC WT10g [2] collection as a testbed for our experiments. Our experiments require result pages from different servers (each indexing different documents), as well as some server selection mechanisms. The WT10g corpus was not necessarily created for conducting DIR experiments. Therefore, we created two different testbeds containing result pages from different servers. The following subsections describe our testbeds and present several server selection mechanisms.

3.1 Result Page Creation

The PF/Tijah retrieval platform was used to create result pages for which each result has a rank, title, summary and URL. PF/Tijah expects its input to be valid XML. Therefore, the first step was to convert all WT10g data into valid XML. We used a program that: 1) discarded the HTML comments, scripts, and all HTML tags except the title and anchor tags; 2) truncated URLs by removing all '/index. . . '-endings, such as /index.html; 3) marked 'sentence-boundaries' in such a way as to create sentences of about 40 to 160 characters. This was done for the purpose of sentence ranking, which is used for creating the document summaries [15]; finally, 4) if a document did not have a title, a title was created from the first sentence of the document.

The second step was to re-group the web pages by their IP-address. This resulted in XML documents containing all web pages from a single server, and we refer to these newly created documents as *ip-grouped* documents. We re-grouped the web pages because we assumed that the pages that make up a website are highly related to each other and that they most often reside on the same web server. Since the web pages in the original WT10g corpus were randomly distributed over several file chunks, we had to perform this additional step.

The third step is to create servers and populate them with the ip-grouped documents. A simple set of rules was used to create these servers. First, we sorted the ip-grouped documents by their file size. Then we selected the smallest ip-grouped document and assigned it to a server *only* if the server was empty or if the server's new size would not exceed a specified size of X MB. Note that an ip-grouped document bigger than X MB was not split. We created two testbeds for our DIR experiments by setting X to 100MB and 500MB. Splitting the WT10g corpus in chunks of roughly 100MB resulted in 79 servers, whereas splitting in chunks of 500MB resulted in 15 servers.

In the fourth and final step, we created an index for each server, and submitted the queries to the servers to obtain the required result pages. These pages contained a maximum of 50 results, and a number indicating the total number of documents found by that server.

3.2 Server Selection

A user is typically only interested in the first N, say 20, results. This means that querying more than N servers wastes valuable resources. In addition, it is not efficient to query a server that will return no relevant results. Therefore, the broker *must* select a small number of the most promising servers.

A results merging method should produce the best possible merged-rankings given any (possibly very poor) set of selected servers. However, we are still far from that ideal. A random server selection or one based on the server's retrieval performance would probably yield significantly different merged-rankings, even in the case where the identical set of servers were selected, albeit in a different order. A server's retrieval performance can be measured by, for instance (4), the Average Precision (AP) measure [17].

$$\text{AP} = \frac{\sum_{i=1}^{N} \text{precision}(i) * \text{rel}(i)}{reldocs} . \quad (4)$$

where precision(i) is the fraction of relevant documents retrieved up to and including rank i; rel(i) is a binary function producing the value 1 when a document at rank i is relevant and 0 otherwise; and $reldocs$ is the number of relevant documents in the document collection for this particular query.

Several server selection strategies are briefly described below.

CORI The CORI server selection algorithm—using the complete (i.e., no QBS) term statistics from each server's index to calculate the CORI-belief score.

Merit A strategy that ranks the servers based on the number of relevant documents in their document collection.

Local-AP A performance-based selection strategy similar to (4), but where $reldocs$ refers to the number of relevant documents in the server's document collection.

Global-AP A performance-based selection strategy similar to (4), but where $reldocs$ refers to the number of relevant documents in the combined document collection of all servers.

4. MERGING APPROACHES

We implemented two SVM learning methods: Ranking-SVM and Regression-SVM. We used Round Robin (RR) and CORI (which was briefly discussed in Section 2.1) as our merging baselines. However, CORI-merging requires the belief scores produced by the CORI-selection schemes; therefore, whenever we use other selection schemes, RR is our only baseline.

The remainder of this section elaborates on the RR and SVM merging approaches.

4.1 Round Robin

Round Robin merging is the simplest merging method and is defined as follows: given n result lists $L_1, L_2, \ldots, L_n$, take the first result r_1 from each list L_i as the first n results. Then take the second result r_2 from each list as the next n results, and so on. RR merging produces a list: $L_1r_1, L_2r_1, \ldots, L_nr_1$, $L_1r_2, L_2r_2, \ldots, L_nr_2$, $L_1r_3, L_2r_3, \ldots, L_nr_3$, etcetera.

Often, the rank of the results is the only feature used when doing RR merging. However, with information about the relevant document distributions of the servers, i.e. the *server score*, we could first rank the servers. By combining both the server score and the result rank, RR can pick the next best result from the next best server, thereby improving its merging performance.

4.2 Learning

This subsection explains the features and labels of the training data for both SVM approaches, and how we validated our models.

4.2.1 Features

Table 1 lists the features used in our experiments. All features are grouped into some category and each category states the number of features between brackets. For example, the second group (Server rank) has one feature which is the score given by one of the four server selection strategies, whereas the final group (Result's term diversity) has three features telling us something about the diversity of the words and characters contained in a given result. The abbreviations LCS, LWO, and LM denote Longest Common Substring, Longest Word Order, and Language Model respectively. The letters q, t, s, f, p, and u stand for query, title, summary, fqdn (Fully Qualified Domain Name), path, and URL (u = f+p), respectively.

LM(a,b) is a simple language model similarity between a and b: the term-frequency statistics are taken only from the text found in b, and a constant of 0.001 is used for smoothing. We also implemented an LM algorithm that allows partial matching (denoted by LM-p). An example of partial matching is when the query 'chair' matches a piece of text such as 'wheelchairs.com'.

LCS(a,b) detects the greatest unaltered proportion of string a that also appears in exactly the same way in b. LWO(a, b) is almost similar to LCS, but it allows for noise. For example, let a denote the text "using ranking SVM in IR" and let b denote "using Machine Learning techniques for ranking in IR". The LCS similarity between a and b is fairly low (0.4), while the LWO similarity yields a score of 0.8.

For a given server, D_{found} denotes the total number of documents found. D_{min} and D_{max} denote the minimum and maximum number of documents respectively found by the selected servers.

We grouped the features for the purpose of feature selection: when we trained a model, we tried different combinations of the feature groups. Note that the result rank feature was used differently in the two SVM approaches. With the linear rank score, Ranking-SVM performed extremely poorly, while it performed much better with the logistic rank score. For Regression-SVM, the effects of the rank features were the other way around, although the logistic feature was not as dramatic for Regression-SVM as the linear feature was for the Ranking-SVM.

Finally, we also experimented with stemmed and stopped versions of the final six feature groups. In later sections, we will append the suffix '-*ws*' to denote that stemming and stopping were used, and the suffix '-*ns*' to denote that stemming and stopping were not used.

4.2.2 Ranking-SVM

Clicks indicate a preferred ranking that should be learned by the Ranking-SVM algorithm. However, we do not have actual click data, so instead we use the TREC relevance judgments. There are important differences between the two. Clicks are binary and convey relative relevance that is based on superficial information supplied by the search engine (e.g., ranks, titles, summaries, and URLs). WT10g TREC judgments are ternary and convey *absolute* relevance: a team of people have actually read the entire document and

Table 1: List of Features

Result rank (1)
$1 - rank/50$ (for Regression-SVM)
$1 - 1/2 * \log(rank)$ (for Ranking-SVM)
Server rank (1)
the normalized server score
Documents found by server (1)
$(D_{found} - D_{min})$ / $(D_{max} - D_{min})$
Server response (1)
LM: q – top10 server results
Digits (20)
number of [1–4]-digit numbers in $\{q, t, s, f, p\}$
Path (1)
the amount of '/'-characters in p
Language model (4)
LM-p: q – {t, s, f, p}
Longest common substring (4)
LCS: q – {t, s, f, p}
Longest word order (4)
LWO: q – {t, s, f, p}
Result consistency (3)
LM-p: t–s, t–u, s–u
Word statistics (10)
number of words in {q, t, s, f, p}
avg. word length in {q, t, s, f, p}
Result's term diversity (3)
total distinct terms / total terms
most frequent term's frequency / total terms
total non-word characters / total characters

rated it as being irrelevant, relevant, or highly relevant.

Furthermore, the assumption that users scan the ranks sequentially from top to bottom allows us to further assume that a higher ranked document that was not clicked is probably less relevant than a lower, clicked, document. This is not the case with TREC judgments; our retrieved documents were not judged in sequential order, so the standard assumption that unjudged documents are irrelevant might lead to learning a sub-optimal ranking function when treating unjudged results as irrelevant. Therefore, we decided to discard the unjudged results when training an SVM model.

As an example of how we used the TREC judgments to create the preference constraints, consider the following rankings where result 2 is irrelevant, results 1 and 5 are relevant, and result 3 is highly relevant. Discarding the unjudged result, the preferred ranking is: 3,1,5,2. The preference constraints are $3 \succ 1$, $3 \succ 2$, and $5 \succ 2$. For each click, Joachims [6] added random additional constraints that should stabilize the learned ranking. We also added 10% (of the total results being merged) of additional random constraints. In addition, we only chose randomly from the set of results that were less relevant than the 'clicked' document; however, this is impossible if you only have clickthrough data.

Finally, we restricted the ranks at which we "observe" the clicks: we only look for clicks within the top 15% of the rankings. For example, in a page with 50 results, if ranks 7 and 8 are relevant, we only create the preference constraints for the result ranked 7^{th}. This restriction led to a substantial gain in the retrieval performance of the learned ranking function.

4.2.3 Regression-SVM

Using Regression-SVM, we aim to predict the *absolute* rank of a given result. This rank should reflect the gathered knowledge from both the TREC judgments as well as of the servers' rankings. However, the TREC judgment should have a higher impact on the learned ranking function. For instance, if a highly relevant result (according to the TREC judgment) was ranked lowest by some search engine, then we certainly want our learned ranking function to rank that result somewhere near the top.

Just as with our Ranking-SVM approach, we excluded unjudged results from our training data in order to avoid unnecessary noise. We label each training instance simply by the value obtained when deducting its rank from either fifty or one hundred, depending on whether the result was irrelevant or not, respectively. The resulting label ensures that all relevant documents (according to the TREC judgments) are ranked in the top positions, followed by the irrelevant documents. Also, within each class of (relevant or irrelevant) documents, the documents are further ordered based on the original rankings of the search servers.

4.2.4 Validation

Our training data consisted of the result pages for the fifty *odd*-numbered queries, taken from a set of N servers. (The queries were taken from TREC topics 451–550.) The servers were selected using selection strategy S. We also varied the set of features F used for training. During training, we used the default values for the SVM-parameters. Each combination of N, S, and F yields a different training set and thus a (potentially) different model. To validate all these models, and choose the model with the best retrieval performance, we used 25-fold cross-validation.

Each fold determines the set of queries QT that will be used for training, and the set QV that will be used for validation. In particular, we focused our validation on merging results from the top 3, 4, and 5 servers. For instance, for each fold, we validated on (QV, 3, S, F), (QV, 4, S, F), and (QV, 5, S, F), and we recorded the averaged Local-MAP and Global-MAP as that fold's validation score.

After cross-validating, we chose the model with the highest Global-MAP, and the one with the highest Local-MAP; this was done for both Ranking-SVM and Regression-SVM. In other words, we selected a total of four models.

5. EVALUATION

We evaluated the different approaches by measuring their Global-MAP when merging the results of the even-numbered queries of the top N servers, which were selected following one of the available server selection strategies.

To test whether the merging methods were significantly (with $p<0.05$) better than the RR or CORI merging method, we

Table 2: Ranking-SVM weights

	SVM-0		SVM-1
result rank	3.544	result rank	3.325
LWO-*ws*(q, t)	-0.557	LWO-*ns*(q, t)	-0.443
LWO-*ws*(q, s)	0.834	LWO-*ns*(q, s)	1.391
LWO-*ws*(q, f)	0.198	LWO-*ns*(q, f)	0.612
LWO-*ws*(q, p)	-0.898	LWO-*ns*(q, p)	0.162

used a randomization approach [14] with 100,000 random permutations. Our test statistic was the Global-MAP of each merging approach.

6. RESULTS

In this section, we present and discuss the performance of the merging methods: CORI, RR, and the four SVM models that were chosen by cross-validation. We will start by discussing the cross-validation results, after which we will discuss the test results.

6.1 Cross-Validation Results

The Ranking-SVM model which has the highest (cross-validated) Local-MAP was trained on the results pages of the top 3 GAP-selected servers, with the result rank and LWO-*ws* features. We will refer to this model as Ranking-SVM-0.

The Ranking-SVM model with the highest Global-MAP was trained using the results pages of the top 3 GAP-selected servers, with the result rank and LWO-*ns* features. We will refer to this model as Ranking-SVM-1.

The Regression-SVM model which has the highest Local-MAP was trained using the results pages of the top 5 GAP-selected servers, and the following features: result rank, server rank, LCS-*ws*, iz-*ns*. We will refer to this model as Regression-SVM-0.

The Regression-SVM model with the highest Global-MAP was trained using the results pages of the top 3 GAP-selected servers, and the following features: result rank, LCS-*ws*, LM-p-*ns*, iz-*ns*. We will refer to this model as Regression-SVM-1.

The learned feature weights of the models can be seen in Tables 2 and 3. As you can see, the result rank feature is the most important feature.

6.2 Test Results

Figures 1, 2, and 3 show how the Global-MAP changes as the number of selected servers increases. Figures 4, 5, and 6 show how the Precision@10 changes as the number of selected servers increases. There is a figure for each server selection strategy and both collections sizes.

In all six figures, the first row of numbers on the x-axis denotes the number of selected servers, while the second row denotes the average number of relevant documents per query, which is a direct consequence of the server selection strategy.

Keep in mind that we want to select as few servers as possible (e.g., to minimize network traffic and computing time), while

Table 3: Regression-SVM weights

	SVM-0		SVM-1
result rank	50.000	result rank	49.142
server rank	0.000		
LCS-*ws*(q, t)	0.000	LCS-*ws*(q, t)	-0.314
LCS-*ws*(q, s)	0.000	LCS-*ws*(q, s)	0.295
LCS-*ws*(q, f)	0.002	LCS-*ws*(q, f)	0.027
LCS-*ws*(q, p)	0.002	LCS-*ws*(q, p)	-0.009
iz-*ns*(q, s)	0.000	iz-*ns*(q, s)	0.062
iz-*ns*(q, f)	0.000	iz-*ns*(q, f)	-0.072
iz-*ns*(q, p)	0.000	iz-*ns*(q, p)	0.044
		LM-*ns*(q, t)	0.132
		LM-*ns*(q, s)	0.139
		LM-*ns*(q, f)	-0.169
		LM-*ns*(q, p)	0.831

at the same time, we want the merging performance to be as high as possible.

When using the LAP and GAP selection strategies, RR is always significantly better than the SVM models. Sometimes, the differences between the SVM models are also significant. When using the CORI selection strategy, from five servers onwards, both CORI and RR are usually significantly better than the SVM models. Keep in mind that when doing multiple comparisons, we would expect some significant differences to actually be false alarms.

6.2.1 CORI selection

Using CORI selection, the retrieval performance of all models is much lower than with any other selection method, as can be seen from the Global-MAP figures as well as the P@10 figures. The performance of both baselines—RR and CORI-merging—is almost indistinguishable.

Compared to LAP-selection, the first few servers selected by CORI-selection contain almost twice as many relevant documents per query on average (the small numbers below the x-axis), yet none of the merging methods seem able to exploit this fact. The extremely poor performance of RR (compared to the other server selection strategies) indicates that CORI-selection often selects servers that return no relevant results at rank one. Furthermore, since no other merging method outperforms RR on this data, it suggests that it is difficult to discriminate between relevant and irrelevant results, at least in this particular set of results.

6.2.2 GAP selection

Using GAP selection, RR clearly outperforms the other merging methods. The margin by which RR outperforms the other models is unexpected, especially since the result's rank seems to be the most important feature for all models (just as for RR), as can be seen in Tables 2 and 3. Note that the range of all feature values lies between one and zero, except for the LM features (of which we have seen values ranging from zero up to five).

7. CONCLUSION

Merging search results from different servers both efficiently and effectively is a major problem in Distributed Information Retrieval.

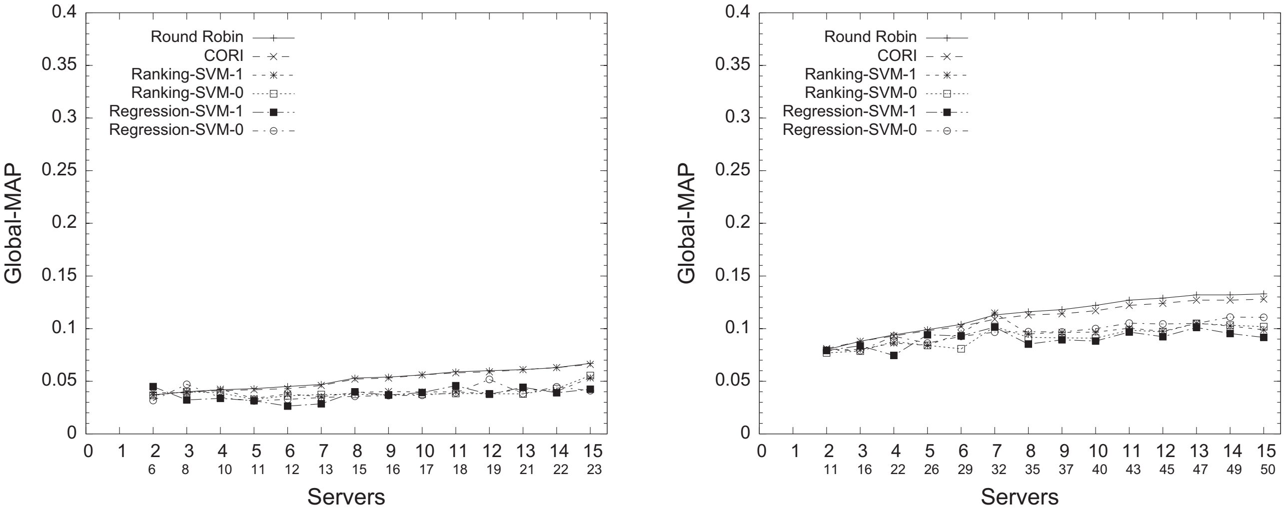

Figure 1: Global-MAP for CORI-selection on the 100MB (left) and 500MB (right) collections

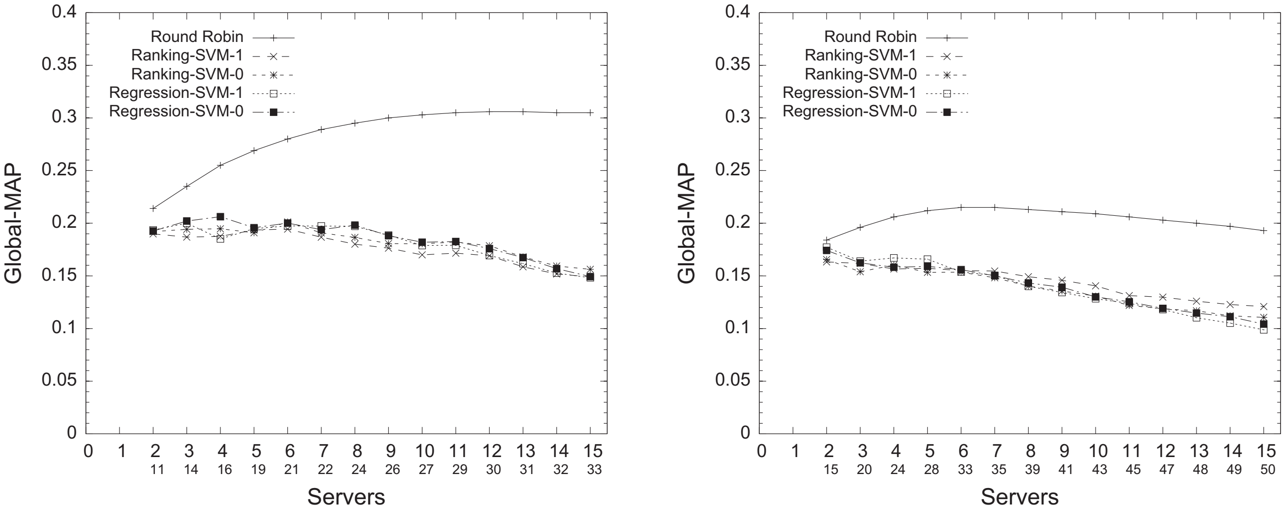

Figure 2: Global-MAP for GAP-selection on the 100MB (left) and 500MB (right) collections

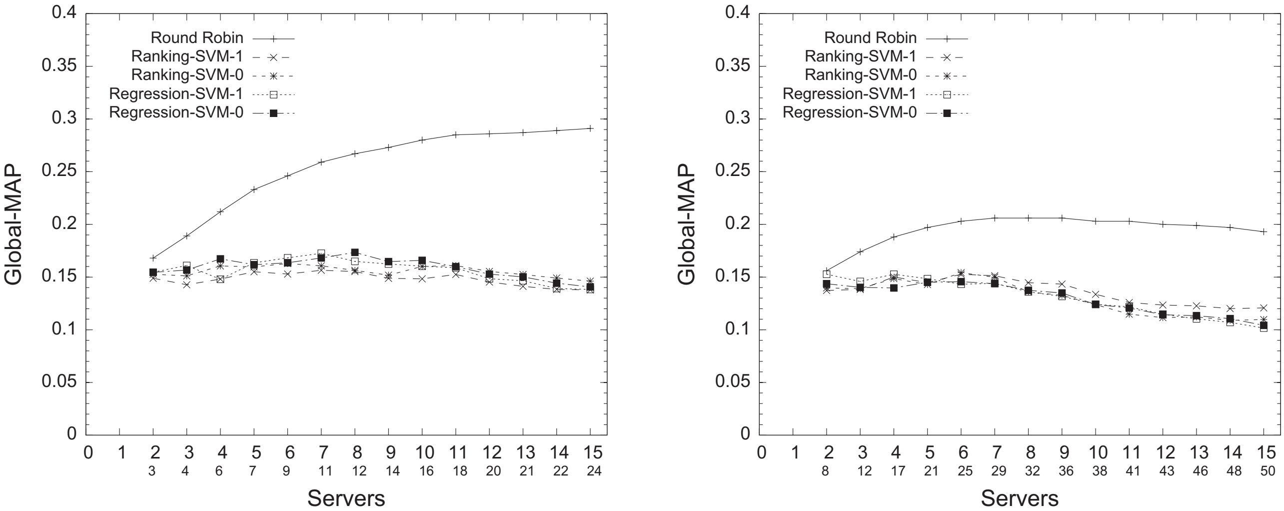

Figure 3: Global-MAP for LAP-selection on the 100MB (left) and 500MB (right) collections

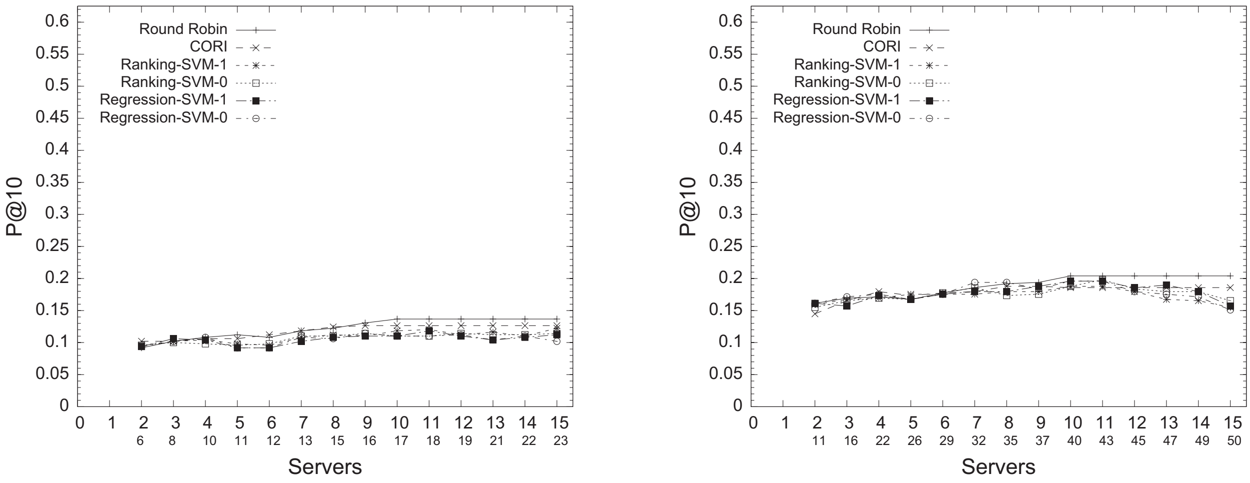

Figure 4: P@10 for CORI-selection on the 100MB (left) and 500MB (right) collections

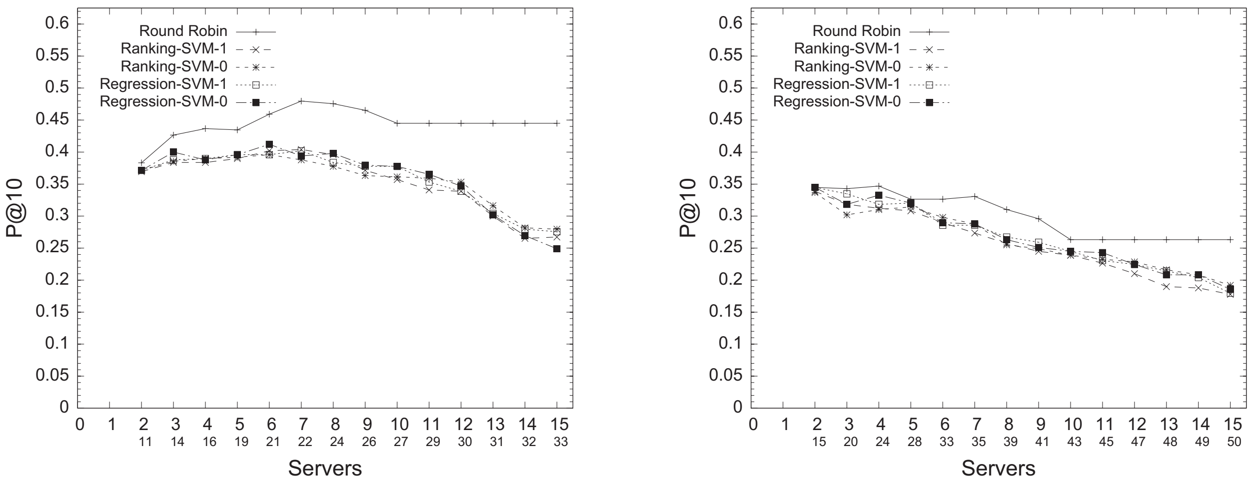

Figure 5: P@10 for GAP-selection on the 100MB (left) and 500MB (right) collections

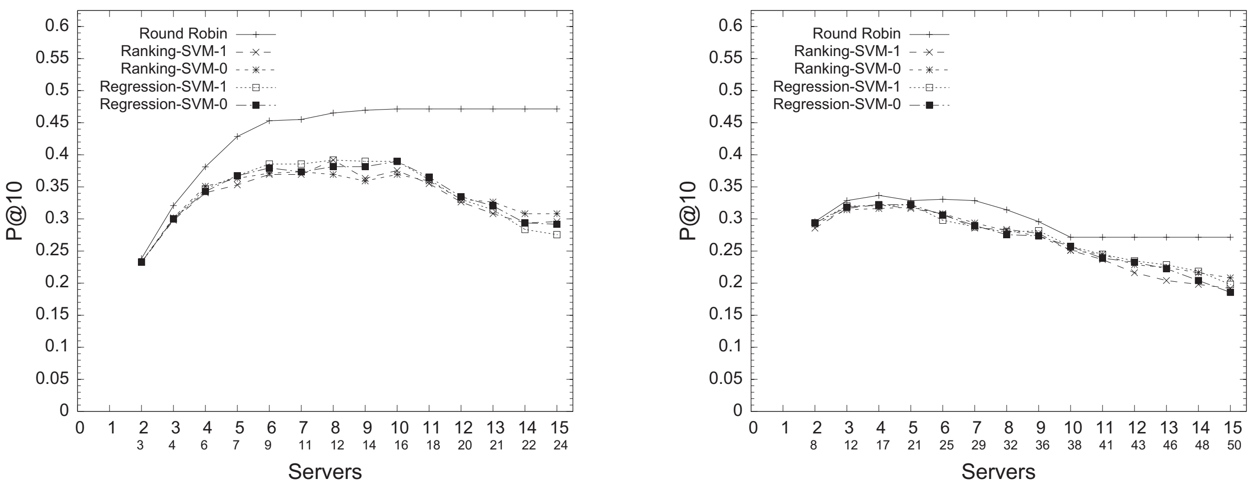

Figure 6: P@10 for LAP-selection on the 100MB (left) and 500MB (right) collections

Our approach avoids the use of document scores and learns a ranking function—using Support Vector Machines—that can merge results based on information that is readily available: i.e. the ranks, titles, summaries and URLs, contained in the result pages. By not downloading additional information, such as the full document, we decrease bandwidth usage.

We have experimented extensively with many different feature combinations to find a good ranking function. We trained a ranking-SVM model that uses pairwise training instances to learn a ranking function, and a regression-SVM model that uses pointwise training instances.

However, our experiments show that the SVM-methods do not improve over the baselines.

8. DISCUSSION

Using Ranking-SVM proved to be very much more sensitive to the type of features used, and the way in which they are preprocessed, as compared to Regression-SVM.

It is disappointing that the SVM approaches were unable to achieve a better performance than Round Robin. One might argue that in real life, no such thing exists as GAP-selection. However, that does not explain why the SVM algorithms apparently learn a mediocre ranking function when trained with exactly these features (i.e., result rank and server rank, as indicated by GAP-selection).

We also experimented with z-normalization for those features that might have a different order of magnitude, depending on the query. Z-normalization works as follows: for a feature f, we compute a new score $s_f^{'} = (s_f - \mu_f)/\sigma_f$, where μ_f is the mean of all values of feature f, and σ_f is the standard deviation of all values of feature f.

Our preliminary results show that this additional normalization does not lead to an improvement of the learned models.

We used a linear kernel for our experiments; therefore, we cannot conclude that our features are insufficient to optimally merge the results. Using a non-linear kernel could lead to a better model. Our motivation for using linear kernels was that Joachims [6] also used linear kernels, and he also used some features that looked similar to the features that we used.

9. REFERENCES

[1] R. A. Baeza-Yates, C. Castillo, F. Junqueira, V. Plachouras, and F. Silvestri. Challenges on distributed web retrieval. In *ICDE*, pages 6–20. IEEE, 2007.

[2] P. Bailey, N. Craswell, and D. Hawking. Engineering a multi-purpose test collection for web retrieval experiments. *Inf. Process. Manage.*, 39(6):853–871, 2003.

[3] J. Callan. *Distributed Information Retrieval*, volume 7 of *The Information Retrieval Series*, chapter Distributed Information Retrieval, pages 127–150. Springer US, 2000.

[4] J. Callan and M. Connell. Query-based sampling of text databases. *ACM Trans. Inf. Syst.*, 19(2):97–130, 2001.

[5] J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *SIGIR '95: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 21–28, New York, NY, USA, 1995. ACM.

[6] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, pages 133–142. ACM, 2002.

[7] G. Paltoglou, M. Salampasis, and M. Satratzemi. Results merging algorithm using multiple regression models. In *ECIR 2007, Rome, Italy*, pages 172–184. Springer, 2007.

[8] Y. Rasolofo, F. Abbaci, and J. Savoy. Approaches to collection selection and results merging for distributed information retrieval. In *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*, pages 191–198, New York, NY, USA, 2001. ACM.

[9] M. Shokouhi. Central-rank-based collection selection in uncooperative distributed information retrieval. In *ECIR 2007, Rome, Italy*, pages 160–172. Springer, 2007.

[10] M. Shokouhi and J. Zobel. Robust result merging using sample-based score estimates. *ACM Trans. Inf. Syst.*, 27(3):1–29, 2009.

[11] L. Si and J. Callan. Relevant document distribution estimation method for resource selection. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 298–305, New York, NY, USA, 2003. ACM.

[12] L. Si and J. Callan. A semisupervised learning method to merge search engine results. *ACM Trans. Inf. Syst.*, 21(4):457–491, 2003.

[13] L. Si, R. Jin, J. Callan, and P. Ogilvie. A language modeling framework for resource selection and results merging. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pages 391–397, New York, NY, USA, 2002. ACM.

[14] M. D. Smucker, J. Allan, and B. Carterette. A comparison of statistical significance tests for information retrieval evaluation. In M. J. Silva, A. H. F. Laender, R. A. Baeza-Yates, D. L. McGuinness, B. Olstad, Ø. H. Olsen, and A. O. Falcão, editors, *CIKM*, pages 623–632. ACM, 2007.

[15] A. Turpin, Y. Tsegay, D. Hawking, and H. E. Williams. Fast generation of result snippets in web search. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 127–134, New York, NY, USA, 2007. ACM.

[16] V. N. Vapnik. *The nature of statistical learning theory.* Springer-Verlag, New York, Inc., 1995.

[17] E. M. Voorhees and D. K. Harman. *TREC: Experiment and Evaluation in Information Retrieval.* MIT Press, 2005.