An Optimization Framework for Merging Multiple Result Lists

Chia-Jung Lee *, Qingyao Ai *, W. Bruce Croft and Daniel Sheldon
College of Information and Computer Sciences, University of Massachusetts Amherst
Amherst, MA, USA
{cjlee, aiqy, croft, sheldon}@cs.umass.edu

ABSTRACT

Developing effective methods for fusing multiple ranked lists of documents is crucial to many applications. Federated web search, for instance, has become a common practice where a query is issued to different verticals and a single ranked list of blended results is created. While federated search is regarded as collection fusion, data fusion techniques aim at improving search coverage and precision by combining multiple search runs on a single document collection. In this paper, we study in depth and extend a neural network-based approach, LambdaMerge [32], for merging results of ranked lists drawn from one (i.e., data fusion) or more (i.e., collection fusion) verticals. The proposed model considers the impact of the quality of documents, ranked lists and verticals for producing the final merged result in an optimization framework. We further investigate the potential of incorporating deep structures into the model with an aim of determining better combinations of different evidence. In the experiments on collection fusion and data fusion, the proposed approach significantly outperforms several standard baselines and state-of-the-art learning-based approaches.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Search and Retrieval

Keywords

Data fusion; collection fusion; learning to merge; deep neural network

1. INTRODUCTION

Developing effective methods for fusing multiple result lists into one is beneficial to many applications such as web-scale federated search. A large body of fusing techniques has been widely studied, and can be broadly categorized into two tasks, namely data fusion and collection fusion. While both tasks seek an optimal blending of results from different lists, collection fusion has a focus on merging results retrieved simultaneously from multiple collections. Collection fusion, often referred to interchangeably as federated search, is preferred over centralized search alternatives for reasons such as efficiency [35]. In addition, federated search systems can search the content of the hidden web without crawling, producing results comprised of answers returned from multiple verticals [35]. Collection fusion techniques are mostly concentrated on approximating comparable document scores from different repositories [9, 38, 36]. In recent years, the TREC federated web search track established reusable test sets from real web search engines of different verticals [13, 14]. Better performing approaches for the result merging task in this track take into account combined signals such as the number of engines containing a document [31] and text similarity [18].

Data fusion, on the other hand, often refers to tasks where the aim is to merge results retrieved from a single document collection. Compared with individual ranking systems, data fusion provides the advantage that the merged list usually has a higher precision and recall [12]. Known as the *chorus effect* [39], many effective fusion methods are based on the premise that documents that are highly ranked in many of the lists are likely to be relevant [1, 2, 17, 24]. Recent developments in data fusion algorithms rely on using training queries to better estimate the probability of relevance using signals such as document ranks and scores [25, 34, 26].

Recent work [32] on results fusion presents the first attempt that uses relevance judgments to directly optimize retrieval performance metrics such as MAP or NDCG. In particular, Sheldon et al [32] merged results from multiple query reformulations to improve search while mitigating risks such as query topic drift. This approach, LambdaMerge (or λ-Merge), can be classified as a data fusion technique since search runs from different reformulations are conducted on the equivalent of a single collection[1]. λ-Merge learns a scoring function to rank documents by combining features indicating document relevance (e.g., retrieval score or rank of a document) with features indicating the quality of the reformulation and its results (e.g., query clarity and drift).

While λ-Merge has demonstrated strong performance in data fusion, it is unclear how an effective adaption to collection fusion can be carried out. In web-scale federated search, it is beneficial to understand the right medium and identify appropriate verticals for a query. System efficiency,

CIKM'15, October 19-23, 2015, Melbourne, VIC, Australia
 ISBN 978-1-4503-3794-6/15/10...$15.00
DOI: http://dx.doi.org/10.1145/2806416.2806489 [* equal contribution].

[1]The industrial level of document indices should be distributed, but experiments conducted in λ-Merge assumes the presence of every document in every list.

for instance, can be improved when a query is only issued to a number of top verticals. Ranking can further be guided by preferring certain verticals among those selected, resulting in better search performance[2]. In this paper, we address the task of result merging and extend λ-Merge in a way that the new model accommodates and considers the impact of drawing results from different verticals. Specifically, the proposed approach learns a scoring function characterized by not only the quality of documents and resources (i.e., ranked lists produced by different search engines), but verticals that contain the most relevant content. The final ranking of blended results is determined by directly optimizing the overall retrieval effectiveness.

Inspired by the recent success of deep structures [5, 16, 20], our model further investigates the potential of combining different ranking evidence using this approach. In particular, we incorporate a deep neural network (DNN) for learning the scoring function. The structure of a DNN is highly simple and generative, and can be applied to most classification and optimization problems given proper design of the loss function in the output layer. We note that the focus of previous studies is mostly around learning representation [15, 20, 29]; we propose to apply deep structured models to the fusion task.

We conduct extensive experiments in a variety of fusion scenarios. Specifically, we use two recent collections from the TREC federated web search track, FedWeb13 and FedWeb14, for testing collection fusion. For data fusion, it is typical to use existing runs submitted to TREC based on which a merged list is created. We experiment with two years of TREC Web Track data, WebTrack09 and WebTrack10, where measures capturing both precision and diversity (e.g., α-NDCG@k) of a ranked list can be evaluated using the sub-topic relevance judgments. The experimental results demonstrate that integrating vertical signals significantly improves retrieval effectiveness for collection fusion. We further show that, for both fusion scenarios, the performance of models with deep structures consistently outperforms standard baselines [17], state-of-the-art approaches [11, 7, 41], and models with shallow structures.

Since λ-Merge focuses on combining results from different query reformulations, features that quantify reformulation quality and topic drift are unavailable in a more general fusion scenario (i.e., only a singe query is present). One contribution of our work is to introduce new effective features that are able to characterize the quality of ranked lists. The features we use are also more semantically informative (e.g., aggregated retrieval scores), compared with the coarse features (e.g., bi-grams or tri-grams) used in other deep structured work [15, 20]. Although the amount of training instances in our experiments is relatively small, the higher-level definition of our features can make learning an effective deep model possible.

After a discussion of related work, we describe in detail the merging framework in Section 3. We then introduce the tested environments along with the features devised for these collections in Section 4. Section 5 shows the results of fusion experiments using a variety of baselines and the proposed models. We discuss and conclude the paper in Section 6.

[2]For instance, vertical *Game* may be preferred to *Entertainment* for query *Mine Craft* although both are deemed relevant.

2. RELATED WORK

This paper presents a merging framework that uses deep structures. This section first discusses general related work on merging multiple result lists, followed by a brief introduction to the applications of deep structures in the information retrieval field.

2.1 Merging Multiple Result Lists

Data Fusion

Methodologies for fusing multiple search results into a single one can be broadly categorized into two families: data fusion and collection fusion. Data fusion refers to tasks where a query is sent to multiple retrieval models that have access to the same document collection, and the rankings from different models are merged into a final single list [12, 39, 42]. One main advantage of data fusion is that often the merged list will have greater relevance than the output of any individual system [3]. Fox and Shaw [17] suggested several combination methods; among these, CombSUM and CombMNZ showed better effectiveness by fusing results based on normalized document scores and the number of systems containing a document. A follow-up work [24] modified CombMNZ by replacing document scores with ranks. Aslam and Montague [1] proposed to merge documents using the probability of relevance estimated for documents at given ranks. Bordafuse [2] also merged documents based on the ranks and no training data is required.

More recent developments in data fusion have laid an emphasis on techniques that use training queries to estimate scores assigned to documents. ProbFuse [25] divides each result list into equal length segments, and uses training queries to estimate the probability that a document returned in a particular segment is relevant. SegFuse [34] takes a similar approach and modifies ProbFuse by allowing variant size of segments. Building on these techniques, SlideFuse [26] introduces a sliding window to estimate relevance probability. A recent cluster-based fusion approach [22] considers inter-document similarities created across the lists to improve merging effectiveness.

Collection Fusion

Collection fusion focuses on a related yet somewhat different task. In particular, queries are issued to document collections that are disjoint or partially overlapped, and the returned results are integrated and merged into a single list [10, 40]. Result merging is a challenging task as different retrieval algorithms may be applied to different collections that may have different lexicon statistics. Earlier approaches to collection fusion [9] used simple heuristics to normalize collection-specific document scores for ranking. More recent methods approximate comparable document scores with higher accuracy. A semi-supervised learning method proposed by Si and Callan [38] trained a regression model for each collection, which is used to map document scores into their global merging scores. Shokouhi and Zobel [36] assumed that the ranking of sampled documents is a subranking of the original collection, and estimated the merging scores by applying curve fitting to the sub-ranking.

Collection fusion and federated search have been often used interchangeably [35]. Web-scale federated search has recently drawn much attention and become a common practice. To promote related research, the TREC federated web

search track established reusable test sets from real world search engines and their results [13, 14]. In a web setting, a query is issued to a selected subset of engines (or verticals); this is regarded as the task of *resource selection*. The *result merging* step then merges the returned results from the selected resources into a single list. We only focus on the result merging task in this paper. In TREC FedWeb 2013, the best performing approach [31] fuses results by adapting the reciprocal rank fusion approach [11]. For FedWeb 2014, the best performing approach was submitted by the ICTNET group [18] where they fused documents by combining signals such as ranks and text similarity.

2.2 Deep Structure Learning

Deep structured models have been successfully applied to many language related applications such as speech recognition (SR), natural language processing (NLP) and information retrieval (IR) [4, 5, 20, 33]. Deep learning techniques can learn high level abstractions from training data which has been shown useful for both classification tasks and semantic analysis. General structures of supervised deep networks include the recurrent neural network (RNN), deep stacking network (DSN), deep neural network (DNN) and convolutional neural network (CNN) [16]. RNN describes a network that recurrently combines former activation states of hidden neurons with current input layer features to predict the next output. It has been widely used in SR and NLP tasks [27, 29], but not as much in IR. Most deep learning applications in IR focus on the generation of semantic representations for queries and documents. Deng et al. [15] first reported positive results with DSN models. The semantic representations extracted with DSN have higher retrieval performance than traditional learning to rank algorithms. Huang et al. [20] developed a DNN based architecture to learn semantic models with clickthrough data. Huang's model, namely the Deep Structured Semantic Model (DSSM), achieved significant improvements compared to many state-of-the-art latent semantic models. Shen et al. [29] further extended Huang's model with a layer of convolution and max-pooling (which is the main idea of CNN) and obtained even better performance than DSSM.

In our work, we choose the general structure of DNN as the network for the fusion task. Compared with other deep learning models, DNN has a structure that is simple and generative. With proper design of the loss function in the output layer, DNN can be applied to most classification and optimization problems, and it is easier to train than DSN. CNN showed better performance than DNN in tasks of latent semantic analysis; however, it does not fit our problem since our training instances consist of features that have a fixed number of dimensions as opposed to a sequential structure. Similar to CNN, RNN is also not appropriate for our task, which leaves DNN as our best choice.

3. MERGING MULTIPLE RESULT LISTS

Developing effective methods for fusing multiple ranked lists of documents is crucial to many applications. Conventionally, unsupervised approaches use retrieval scores or ranks to determine the final ranking of documents in a merged list. These methods have merits that they require no training examples. It is, however, often the case that learning-based approaches achieve better effectiveness, partly because additional features are easy to incorporate and customized objectives can be derived and optimized. This section first introduces an existing learning-to-merge framework, λ-Merge. We then describe our extension by introducing vertical estimation and deep structures.

3.1 Preliminaries of λ-Merge

The λ-Merge approach is the first proposed merging method that is trained using relevance judgments to directly optimize retrieval metrics such as NDCG or MAP [32]. λ-Merge considers, for a query q, a set of reformulations $\{q_1, q_2, \cdots, q_N\}$ (including the original q), and the goal is to fuse the corresponding output result lists $\{D_1, D_2, \cdots, D_N\}$ into a single one. For each document d in the i^{th} result list, λ-Merge utilizes multiple *query-document* features x_i^d to capture how relevant d is to query q_i. In addition, for the i^{th} result list returned of the i^{th} reformulation, a vector of *query-list* features z_i is created to estimate the quality of that ranked list. These query-list features mostly focused on characterizing each query reformulation's difficulty and drift, including features such as query clarity score or the distance between original and rewrite queries (e.g., random walk probabilities between queries).

The core components of λ-Merge consist of a scoring function $f(x_i^d;\theta)$ and a gating function $g(z_i;\eta)$, based on which the parameters θ for features x_i^d as well as η for features z_i can be learned. The model defines the final score of a document d by re-weighting $f(x_i^d;\theta)$ with the gating component as in Equation 1, where L is the total number of lists.

$$s_d = \sum_{i=1}^{L} g_i \cdot f(x_i^d;\theta) \tag{1}$$

In the implementation of λ-Merge, the scoring function $f(x_i^d;\theta)$ is carried out using a fully connected neural network with a *single* hidden layer of four neurons, each having a *tanh* activation function. The hidden units are connected to an output unit, which is then re-weighted by the linear gating function g_i. The objective of λ-Merge is to produce a single fused list with NDCG optimized. The training is done by taking partial derivatives of s_d with respect to the network parameters θ and η.

3.2 The Proposed Framework

Now we introduce our extensions of λ-Merge. To apply λ-Merge to collection fusion, we incorporate an extra component that captures the quality of a vertical into the framework. The idea is that the quality of a vertical in which a document exists can affect the overall relevance of this document to a query. Figure 1 shows the structure of our framework. Suppose that, for a query q, we have L search engines drawn from a set of V verticals, each search engine producing one ranked list in response to q. Typically we have $V < L$, meaning that multiple engines are adopted for a vertical. The framework in Figure 1 first considers the feature function $f(x_{i,j}^d;\theta)$ derived for a document d that is in the i^{th} ranked list belonging to the j^{th} vertical. The model then re-weights f by g_i, which is defined as the *softmax* output of the feature function $g(z_i;\eta)$ that measures the quality of a ranked list. Finally, we incorporate the estimation for vertical quality by combining $g_i \cdot f$ with h_j. The weight h_j stands for the *softmax* output of feature function $h(y_j;\phi)$ that captures the goodness of a vertical given the current

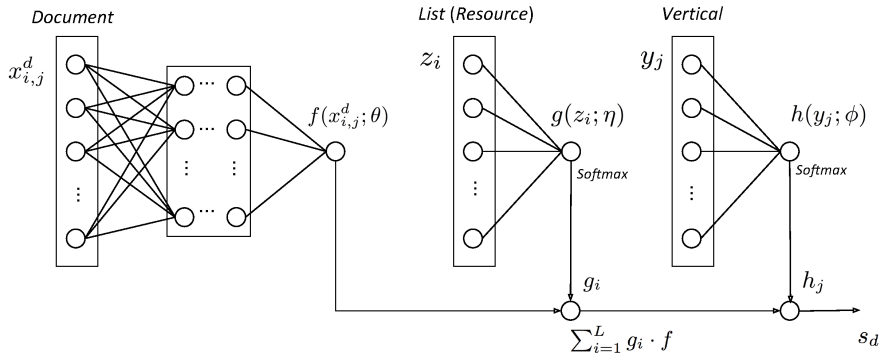

Figure 1: The proposed framework for merging results from multiple lists. The final score of a document is determined by the document, list, and vertical feature functions based on a DNN structure.

search context. Overall, the model produces the final document score s_d in the merged list using Equation 2.

$$s_d = \sum_{j=1}^{V} h_j\big(\sum_{i=1}^{L} g_i \cdot f(x^d_{i,j};\theta)\big) \tag{2}$$

Inspired by the widely reported effectiveness of deep models, our second extension investigates the potential of incorporating multiple hidden layers in an artificial neural network for the fusion task. We take the definition of deep structures to include properties that (1.) multiple layers of nonlinear processing units are incorporated and (2.) the supervised or unsupervised learning of feature representations takes place in each layer, with the layers forming a hierarchy from low-level to high-level features [16]. As shown in Figure 1, the implementation of the feature function $f(x^d_{i,j};\theta)$ is based on a DNN structure.

Feature functions for scoring

We implement the scoring function $f(x^d_{i,j};\theta)$ using a standard feed-forward artificial neural network that incorporates two or more layers of hidden neurons. Formally, denoting l_k as the input of intermediate hidden layer k and o as the output of neural network, the projection process can be described as in Equation 3. Here W_k and b_k are the projection matrix and bias term for hidden layer k, and a denotes the activation function. The parameter space is therefore $\theta = \{W_k, b_k : \forall k\}$.

$$\begin{aligned} l_1 &= W_1 x^d_{i,j} \\ l_k &= a(W_k l_{k-1} + b_k), k = 2, 3, ..., N-1 \\ o &= a(W_N l_{N-1} + b_N) \end{aligned} \tag{3}$$

We test the activation function a with both *tanh* and *sigmoid*. In practice, no substantial difference was observed and we opt for *tanh* in each neuron:

$$a(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{4}$$

Similar to the implementation of f, we design a neural network structure separately for the list feature function $g(z_i;\eta) = \eta^T z_i$ and the vertical feature function $h(y_j;\phi) = \phi^T y_j$. The two networks both follow a standard feed-forward propagation, and respectively taking query-list features z_i and query-vertical features y_j as input layer signals. To determine the re-weighting component g_i, we propagate $g(z_i;\eta)$ through a *softmax* smoothing function as shown in Equation 5, where L denotes the number of input rank lists. The same is done for h_j using Equation 6. In theory, these two weighting networks can incorporate deep structures as in the document scoring network $f(x^d_{i,j};\theta)$. This will require considerably more training data since list and vertical features are sparser than document ones; we thus choose to leave it as a shallow network.

$$g_i = \frac{exp(g(z_i;\eta))}{\sum_{k=1}^{L} exp(g(z_k;\eta))} \tag{5}$$

$$h_j = \frac{exp(h(y_j;\phi))}{\sum_{k=1}^{V} exp(h(y_k;\phi))} \tag{6}$$

Optimizing retrieval performance

The result merging task seeks a single ranked list by fusing multiple ones with an aim of achieving retrieval effectiveness as high as possible. Accordingly, we train the model parameters θ, η and ϕ to directly optimize NDCG@20 of the fused list using a gradient-based approach. We note that other measures such as MAP or Precision@k can be used.

A problem that arises with the direct optimization of search effectiveness is that the retrieval metrics are discontinuous with respect to document scores[3]. To overcome the discontinuity of the objective, Burges et al [6] suggests it is sufficient to define the λ functions for which some objective C exists (as opposed to defining C directly). This technique is also used by the λ-Merge model. In our framework, it is sufficient to learn the model by implementing the gradients with respect to the model parameters θ, η and ϕ; that is, our goals are to compute $\frac{\partial C}{\partial \theta_k}$, $\frac{\partial C}{\partial \eta_k}$ and $\frac{\partial C}{\partial \phi_k}$.

The computation for these gradients can be decomposed using chain rule. Take θ_k as an example; we compute $\frac{\partial C}{\partial \theta_k} = \frac{\partial C}{\partial s_d} \cdot \frac{\partial s_d}{\partial \theta_k}$ where the index d is summed over. The former part $\frac{\partial C}{\partial s_d}$ can be solved by the λ functions in LambdaRank [8] as in Equation 7. Here, $d \succ j$ denotes that d has a larger relevance label than j, meaning that d should be ranked

[3]Metrics are determined by the ranks sorted by scores rather than the actual scores.

above j. $|\Delta_{NDCG@20}|$ denotes the variation in NDCG@20 when the two documents d and j are swapped.

$$\frac{\partial C}{\partial s_d} = \sum_{j;d\succ j} \frac{-1}{1+e^{(s_d - s_j)}} |\Delta_{NDCG@20}| \quad (7)$$

To solve $\frac{\partial C}{\partial \theta_k}$, we further need to calculate the latter part $\frac{\partial s_d}{\partial \theta_k}$ of the decomposition. Taking the fact that only the document scoring network involves θ, the gradients can be done as in Equation 8. The update process of $\frac{\partial f(x^d_{i,j};\theta)}{\partial \theta_k}$ follows the general algorithm of back-propagation (BP).

$$\frac{\partial C}{\partial \theta_k} = \frac{\partial C}{\partial s_d} \cdot \frac{\partial s_d}{\partial \theta_k} = \frac{\partial C}{\partial s_d} \cdot \sum_{j=1}^{V} h_j \big(\sum_{i=1}^{L} g_i \cdot \frac{\partial f(x^d_{i,j};\theta)}{\partial \theta_k} \big) \quad (8)$$

Likewise, for $\frac{\partial C}{\partial \eta_k}$ and $\frac{\partial C}{\partial \phi_k}$, we need to solve $\frac{\partial s_d}{\partial \eta_k}$ and $\frac{\partial s_d}{\partial \phi_k}$ as we already know $\frac{\partial C}{\partial s_d}$. The computation can be done as in Equations 9 and 10.

$$\frac{\partial s_d}{\partial \eta_k} = \sum_{j=1}^{V} h_j \big(\sum_{i=1}^{L} \frac{\partial g_i}{\partial \eta_k} \cdot f(x^d_{i,j};\theta) \big) \quad (9)$$

$$\frac{\partial s_d}{\partial \phi_k} = \sum_{j=1}^{V} \frac{\partial h_j}{\partial \phi_k} \big(\sum_{i=1}^{L} g_i \cdot f(x^d_{i,j};\theta) \big) \quad (10)$$

Since the output of re-weighting network $g(z_i;\eta)$ goes through a *softmax* smoothing before being incorporated into the model, the partial differential equations are transformed according to Equation 11, where $\frac{\partial g(z_i;\eta)}{\partial \eta_k}$ can be inferred through the process of BP. Similar computation can be done for $\frac{\partial h_j}{\partial \phi_k}$ by replacing Equation 11 with the corresponding parameters.

$$\frac{\partial g_i}{\partial \eta_k} = \frac{\beta_i \frac{\partial g(z_i;\eta)}{\partial \eta_k} \sum_{j=1}^{L} \beta_j - \beta_i \sum_{j=1}^{L} \beta_j \frac{\partial g(z_j;\eta)}{\partial \eta_k}}{(\sum_{j=1}^{L} \beta_j)^2} \quad (11)$$
$$\beta_i = exp(g(z_i;\eta))$$

4. TEST BEDS AND FEATURES

Our experiments are conducted in both collection and data fusion environments. This section describes the selected test sets for each environment, and discusses the features in detail.

4.1 Collection Fusion Test Sets

Collection fusion refers to search tasks where a query is issued to disjoint or low overlap document collections, and a single ranked list of blended results is created. To this end, we use test sets derived from the TREC federated web search track in years 2013 and 2014. The main reason for using FedWeb collections is that the track provides test sets in a more realistic web setting. Specifically, the documents were collected from real web search engines as opposed to artificial ones such as those created by dividing existing TREC collections by topic or source [19]. The test sets contain the actual results of approximately 150 real web search engines drawn from a fixed set of 24 verticals, each providing their own retrieval method and heterogeneous content types such as news, travel, social etc. In each year, 50 official queries for the result merging task were created, and up to 10 ranked results were crawled and stored from each of the search engines[4]. We note that only the ranks are stored and no score information is available. Graded relevance judgments are gathered for these queries, following the conventions used in TREC web tracks, in 5 levels {Nav, Key, HRel, Rel, NRel}. The main evaluation metric used in the FedWeb track is NDCG@20.

For both years of data, one characteristic of the document collections is that search engines can return duplicate documents, although stored as different document identifiers. This can happen even within a search engine, meaning that this engine unexpectedly returned the same documents. To prevent rewarding merged search results containing duplicate (relevant) content, we pre-process the document collections using two steps. First, for each ranked list, we keep only one copy of a document and discard any subsequent duplicates. The remaining documents are sequentially moved to lower ranks following the original order. After the first pass of processing, we check if duplicates exist across different ranked lists for a given query. We assign the inter-list duplicates the same document identifier so as to let the merging framework know that they are actually the same. The official lists of duplicate documents can be obtained from TREC FedWeb13 and FedWeb14 sites, which were produced based on a number of heuristics such as identical smoothed URLs.

4.2 Data Fusion Test Sets

Data fusion techniques combine the ranked lists of multiple document retrieval systems to improve search coverage and precision. Different from collection fusion, the retrieval systems operate on the same set of documents in general, and thus the overlap rate across lists is much higher. Conventionally, existing runs submitted to TREC are used to evaluate this task. We experiment with two years of TREC Web Track data, WebTrack09 and WebTrack10. Using these newer data sets for experiments provides the advantage that we are able to evaluate not only precision but also the diversity of a ranked list based on the sub-topic relevance judgments. The test set of each year consists of 50 queries; 71 and 56 runs were submitted to 2009 and 2010 respectively. We randomly sampled 30 ranked lists from each year for the merging experiments.

4.3 Features

Our framework inherits the property of λ-Merge where features approximating document relevance and list quality are employed together for predicting document scores. We further incorporate vertical features in addition to those two categories. Table 1 shows an overview of the feature set used for the experiments, which we describe in detail below.

Document features.

Denoted as $x^d_{i,j}$, document features are used to describe the relation between a query and a document. In Table 1, the rank features correspond to positions where a document is assigned in a list. Reciprocal rank score $1/(60 + rank)$ is incorporated for its reported effectiveness [11].

Typically a document should be promoted in the final ranked list if more search systems deem it relevant. As such,

[4]FedWeb13 stores both snippets and documents of results while FedWeb14 only stores snippets

Table 1: Document, list and vertical features used for learning to merge results. Partial features are only applicable to either FedWeb (F) or WebTrack (W). N denotes the total number of documents in a result list.

Feature	Descriptions	Avail
query-document feature		
Rank	rank, 1/(60+rank)	F, W
Co-exist	number of lists a document exists	F, W
Exist	exist-within/original-length	F
Score	retrieval scores, normalized retrieval scores, weighted scores	F, W
query-list feature		
mCo-exist	mean of document co-exist scores of entire list	F, W
mCo-exist-k	mean of document co-exist scores of top k results, $k \in \{10, 20, 30\}$	W
mScore	mean of document (normalized) scores	F, W
Ratio-1	ratio of documents with co-exist > 1 to N	F, W
Ratio-k%	ratio of docs with co-exist $> k$% of retrieval systems to N, $k \in \{20, 40, 60, 80\}$	W
RatioRet	ratio of number of returned results to number requested	F
RatioDup	ratio of number of remained documents after de-duplication to number returned	F
query-vertical feature		
vmScore	mean of mScore of the ranked lists a vertical has for a query	F
vmCo-exist	mean of mCo-exist of the ranked lists a vertical has for a query	F
vRatio-1	mean of Ratio-1 of the ranked lists a vertical has for a query	F

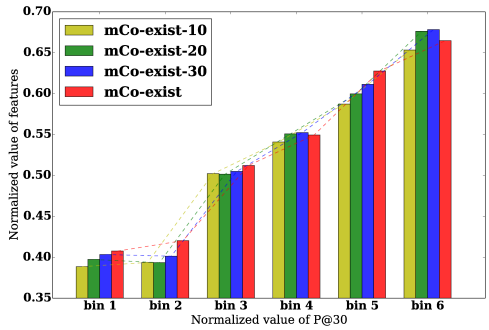

Figure 2: Relation between aggregated document co-exist scores and list performance using WebTrack09.

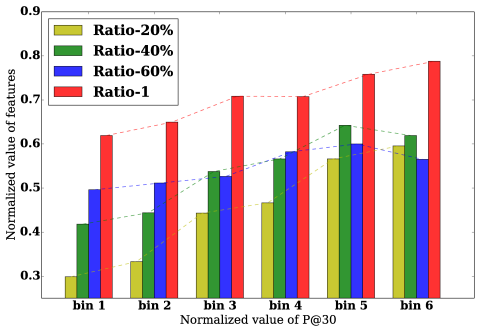

Figure 3: Relation between list ratio scores and list performance using WebTrack09.

the *co-exist* feature counts the number of result lists (i.e., search systems) in which a document appears. In the special case of FedWeb test sets, we also consider the property that a document can appear more than once in a list. The number of duplicates of document in a list divided by the original list length is included as a feature, which is denoted as the *exist* feature.

The score features reflect how relevant a ranker believes a document is to a query. We use two retrieval systems to estimate the score, including a query likelihood model and a sequential dependency model [28]. We also consider the zero-one normalized scores, and the weighted scores computed by taking products of *co-exist* and normalized scores. Finally, the same computation applies to scores from runs submitted to TREC when available.

List features.

Since the focus of λ-Merge is to combine results from different query reformulations, the emphasis for list features is mostly around the reformulation quality and its drift from original query, which is seldom available in a more general fusion setting. To propose more features that may be useful for quantifying list quality, we conduct a preliminary analysis on the retrieval effectiveness of different ranked lists. We hypothesize that better effectiveness of a list is related to whether the documents in that list are also returned by other search systems frequently. In particular, we compute the average *co-exist* scores of documents in a list, and plots its relation to the retrieval performance of the corresponding lists, as shown in Figure 2. The x-axis is constructed by bucketing the normalized P@30 scores for all ranked lists, and the y-axis shows the corresponding average *co-exist* scores for lists in that bin. The same is done for the top k results, where $k \in \{10, 20, 30\}$. Figure 2 shows consistently strong evidence that better performing ranked lists usually have more documents that appear across multiple lists. We accordingly incorporate *mCo-exist* and *mCo-exist-k* into the query-list features set. Note that we compute *mCo-exist-k* only for WebTrack since a ranked list from FedWeb only has up to 10 results.

Based on a similar idea, we correlate the percentage of documents that appear in more than one search engine (i.e., documents with *co-exist* > 1) with the performance of ranked

lists, as shown in Figure 3. We observe that retrieval performance is higher when a larger set of documents appear in more than one ranked list (see *Ratio-1*). We further inspect how the trend changes w.r.t. the coverage over search systems. That is, in addition to examining documents with *co-exist* > 1, we conduct the same analysis for *co-exist* larger than k percent of total number of search systems (e.g., we count documents with *co-exist* > 6 given 30 ranked lists and $k = 20\%$). Compared with *Ratio-1*, *Ratio-k*% in general shows a similar trend but the gaps between performance buckets appear larger, indicating a higher potential of distinguishing ranked list quality. When k grows to 40% or 60%, we see it is not always true that best performing lists have the largest *Ratio-k*%. This may imply that the list performance, while being highly positively correlated, is not solely determined by co-existence.

We adopt the mean score features (i.e., *mScore*) as in the λ-Merge work[5]. Lastly, for the FedWeb test sets, we include two additional features for quantifying the quality of ranked lists. Although FedWeb was created by requesting 10 results per search engine, there are cases where less than required were returned, which we hope to capture by the *RatioRet* feature. Recall that duplicates can be returned by the same engine for a query. *RatioDup* computes the ratio of number of remained documents after de-duplication to the number returned. These two features are designed to reflect the stability of the black box search engines.

Vertical features.

Intuitively, the quality of a vertical depends on the quality of the ranked lists it contains for a query. We accordingly devise features that take the average of list features across the ranked lists a vertical includes. Deriving vertical features is highly related to the task of vertical selection introduced in FewdWeb 2014 [14], where the goal is to classify each query into a fixed set of 24 verticals. The top performing systems of this task assembled together the results of different techniques; for example, the similarity of vertical and query terms [18] and matching WordNet synonyms for queries and verticals [21]. As our focus is to investigate the utility of incorporating vertical estimates rather than exhausting all possible features, we leave more exploration of feature design as future work.

5. EXPERIMENTS

This section first introduces the setup for the experiments, followed by a detailed presentation of evaluation results tested under a variety of scenarios.

5.1 Experimental Setup and Baselines

Our experiments are conducted using four TREC collections as mentioned in Section 4. In each test case, a total number of 50 queries with associated graded relevance judgments are available. When learning is needed, we split training and test data into 5 folds and perform cross-validation. We evaluate using NDCG@K, as well as α-NDCG@k when sub-topic relevance judgments are available. We train all neural-based models for 25 epochs with learning rate $5{\cdot}10^{-3}$. Pairwise t-test is conducted when appropriate.

[5]Other moments such as variance, skewness or kurtosis could be considered; we excluded those due to lower effectiveness in our experiments.

We compare the results of our framework with three categories of baselines:

- *Score fusion.* CombSUM and CombMNZ proposed by Fox and Shaw [17] serve as standard baselines with which new algorithms are compared [11, 22]. We consider CombMNZ as our baseline since it often outperforms CombSUM. Equation 12 computes CombMNZ that considers the product of the number of lists containing d and the summation of normalized scores from each ranking.

$$CombMNZ = |\{k : d \in D_k\}| \cdot \sum_{k:d \in D_k} NScore(d) \quad (12)$$

- *Rank fusion.* Cormack et al. [11] proposed a simple and yet very effective method, reciprocal rank fusion (RRF), for combing searches. RRF sorts the documents using the formula given in Equation 13, where $c = 60$ is fixed during their pilot investigation. Simple as is, RRF was shown to invariably improve the best of the combined results (including runs with learning-to-rank methods), and outperformed established meta-ranking standards Condorcet Fuse [30] and CombMNZ. The best performing approach in FedWeb 2013 [31] is based on modifying RRF with logarithm and square transformation. We use RRF as a strong baseline.

$$RRFscore = \sum_{k:d \in D_k} \frac{1}{c + rank(d : D_k)} \quad (13)$$

- *Learning to rank.* The merging problem can also be cast as a learning to rank problem. A general LTR setting requires one representation per document; however, in a merging environment, documents can appear as several copies in different ranked lists. To address this, we transform the feature representation from the fusion task by taking the average of feature values for documents that have several copies. With the transformed feature vector, we can easily use any existing LTR method to sort the documents. In particular, we adopt the original gradient decent approach, RankNet [7], and its state-of-the-art variant, LambdaMART [41], in our experiments.

5.2 Performance on TREC FedWeb

Table 2 shows the retrieval performance tested on the FedWeb 2013 and 2014 collections. We denote our approach as LTM-Deep for that it is essentially a framework for learning to merge using deep structures. In each year's data, we compare the proposed framework with a number of baseline approaches, including two runs of CombMNZ that are generated based on normalized query likelihood or sequential dependency model [28] scores[6]. Among the baselines, we see that RRF is very competitive, even though its core idea is rather simple and it requires no training examples. The LTR group tends to perform better in NDCG@20, which is both the main evaluation metric in the track and the chosen optimization metric. In general, LambdaMART appears more effective than RankNet.

[6]Recall that no score information is available in FedWeb; we use two retrieval systems as alternatives.

Table 2: Performance of tested techniques on FedWeb 2013 and FedWeb 2014 test sets. The source includes document (D), list (L) and vertical (V) features. The best results are marked bold. The improvement percentage and pairwise t-test are applied when comparing LTM-Deep (D+L+V) to the rest of approaches († is denoted for p-value < 0.05).

TREC FedWeb 2013					
Source	Method	NDCG@10	NDCG@20	$\Delta_{N@10\%}$	$\Delta_{N@20\%}$
D	CombMNZ-QL	0.545†	0.476†	7.38%	20.49%
	CombMNZ-SD	0.547†	0.475†	6.89%	20.84%
	RRF	0.572	0.508†	2.27%	12.95%
D+L	LTR-RankNet	0.526†	0.483†	11.28%	18.77%
	LTR-LambdaMART	0.515†	0.491†	13.59%	17.00%
	LTM-Shallow	0.553†	0.567	5.88%	1.31%
	LTM-Deep	0.565	0.573	3.54%	0.17%
D+L+V	LTR-RankNet	0.570	0.573	2.63%	0.17%
	LTR-LambdaMART	0.556†	0.572	5.22%	0.35%
	LTM-Shallow	0.574	0.572	1.88%	0.31%
	LTM-Deep	**0.585**	**0.574**	-	-
TREC FedWeb 2014					
Source	Method	NDCG@10	NDCG@20	$\Delta_{N@10\%}$	$\Delta_{N@20\%}$
D	CombMNZ-QL	0.577†	0.485†	8.63%	23.17%
	CombMNZ-SD	0.580†	0.481†	8.05%	24.25%
	RRF	0.594†	0.504†	5.48%	18.50%
D+L	LTR-RankNet	0.554†	0.503†	13.26%	18.74%
	LTR-LambdaMART	0.550†	0.515†	13.92%	15.85%
	LTM-Shallow	0.583†	0.574†	7.57%	4.08%
	LTM-Deep	0.599†	0.574†	4.76%	4.08%
D+L+V	LTR-RankNet	0.601†	0.572†	4.33%	4.37%
	LTR-LambdaMART	0.602	0.576	4.15%	3.65%
	LTM-Shallow	0.624	0.565†	0.56%	5.65%
	LTM-Deep	**0.627**	**0.597**	-	-

For LTM-Deep in FedWeb 2014, the results suggest that incorporating vertical signals (i.e., D+L+V) improves significantly the performance compared to that without (i.e., D+L). The trend is similar in FedWeb 2013. For other learning-based approaches, including the LTR group and LTM-Shallow, it is clear that the model with vertical estimate consistently achieves better performance, confirming our hypothesis that vertical quality helps improve the fusion task.

LTM-Deep further outperforms LTM-Shallow in most cases, and the improvement in NDCG@20 is significant in FedWeb 2014. Despite the scarcity of training data, the results show that merging ranked lists with deep structures can be a promising direction, and we expect the improvement will be more evident with more training data. Overall, LTM-Deep shows the best performance across different baselines, measures and collections.

5.3 Performance on TREC WebTrack

The experimental results for WebTrack 2009 and 2010 are shown in Table 3. In addition to NDCG@k, we evaluate the results using the diversity measure α-NDCG@k based on sub-topic relevance judgments. The vertical component is omitted for data fusion experiments since all ranked lists are from the same vertical. Here, CombMNZ is generated using the normalized scores from runs submitted to TREC. In WebTrack09, RRF and LambdaMART again demonstrate better effectiveness among the baselines, while the RankNet method also appears highly effective in terms of precision.

The LTM group demonstrates overall the most effective results compared with the standard baselines and the LTR group. The LTM-Deep approach further achieves better performance than LTM-Shallow in terms of precision and diversity for both years. The improvement is significant in Web Track 2010. Similar to the results in collection fusion, LTM-Deep shows the best performance across different baselines, measures and collections.

5.4 Discussions

Model Design. The framework in this paper models simultaneously the impact of the quality of documents, ranked lists and verticals[7] for result merging. The experimental results suggest that modeling the diverse aspects of federated search improves ranking in general. The design may further include the features derived from specific types of documents. Tweets, for instance, can be extracted with features that represent recency or popularity, while for a CQA posting, it can be helpful to include authority signals such as the overall rating of an answer. In such cases, the document scoring network $f(x^k_{i,j}; \theta, \pi_1, \pi_2, ..., \pi_N)$ will be paramterized on θ for shared features (e.g., ranks and scores) and π_i for non-shared features, where N indicates the number of types of documents. Since the FedWeb test collections were compiled to conform to a uniform style, it will be necessary to get additional data that describes the characteristics of different types of documents for evaluating this design. As

[7]We note that vertical features are only available in a federated search scenario.

Table 3: Performance of tested techniques on Web Track 2009 and Web Track 2010 test sets. The source includes document (D) abd list (L) features. The best results are marked bold. The improvement percentage and pairwise t-test are applied when comparing LTM-Deep to the rest of approaches († is denoted for p-value < 0.05).

TREC Web Track 2009							
Source	Method	NDCG@10	NDCG@20	α-NDCG@10	α-NDCG@20	$\Delta_{N@20\%}$	$\Delta_{\alpha-N@20\%}$
D	CombMNZ	0.309†	0.288†	0.201†	0.227†	20.22%	22.47%
	RRF	0.337†	0.323†	0.229†	0.263†	7.05%	5.70%
D+L	LTR-RankNet	0.324†	0.309†	0.215†	0.243†	12.05%	14.40%
	LTR-LambdaMART	0.314†	0.307†	0.240	**0.290**	12.59%	-4.14%
	LTM-Shallow	0.352	0.340	0.238	0.274	1.88%	1.46%
	LTM-Deep	**0.360**	**0.346**	**0.243**	0.278	-	-
TREC Web Track 2010							
Source	Method	NDCG@10	NDCG@20	α-NDCG@10	α-NDCG@20	$\Delta_{N@20\%}$	$\Delta_{\alpha-N@20\%}$
D	CombMNZ	0.289†	0.281†	0.415†	0.451†	10.83%	5.76%
	RRF	0.295†	0.296†	0.428†	0.465	5.17%	2.58%
D+L	LTR-RankNet	0.306	0.299	0.304†	0.340†	39.4%	40.29%
	LTR-LambdaMART	0.289†	0.285†	0.404†	0.448†	9.20%	6.47%
	LTM-Shallow	0.301	0.299†	0.421†	0.452†	4.15%	5.53%
	LTM-Deep	**0.312**	**0.311**	**0.443**	**0.477**	-	-

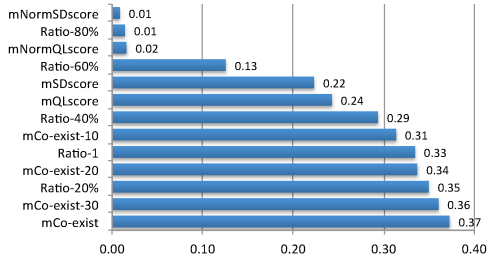

Figure 4: Pearson correlation coefficient ρ between list goodness and *query-list* features on WebTrack10.

aforementioned, the deep structures are also possible in the re-weighting networks for list and vertical features. This will require obtaining additional training data.

Feature Design. One reason behind the success of our framework, we believe, is the introduction of additional query-list features. As mentioned in [32], query-list features such as the mean and standard deviation of retrieval scores showed only small improvements, while the more effective features (e.g., reformulation scores) are not available in a general fusion task. Indeed, computing the Pearson correlation coefficient between feature values and list goodness[8], Figure 4 shows that the newly introduced features usually have a higher correlation. In addition, applying the same correlation analysis between document labels and query-document features, we re-confirm that the *co-exist* feature is the most effective in the data fusion environment. For the collection fusion environments, *co-exist* is far less correlated with ground-truth since the overlap rate between collections is approximately only 3% (as reported in [31]). Interestingly, although *co-exist* independently is not effective enough, the aggregated query-list feature *mCo-exist* is highly correlated with list goodness with $\rho = 0.7$ in both FedWeb 2013 and 2014.

[8]We use the average of document labels in a list as a surrogate to approximate list goodness.

Effective features are critical to system performance. We may explore the feasibility of incorporating the findings from the large body of literature on query performance prediction. In particular, post-retrieval predictors attempt to predict the performance of a ranked list using strategies such as measuring the divergence between the list and the entire collection [23, 37]. Again, since the FedWeb collections use search engines as a black box service, additional access to the entire collection statistics may be required.

6. CONCLUSIONS

In this paper, we explored the task of result merging in an optimization framework. The proposed model incorporated vertical signals on the top of the feature functions for documents and ranked lists, and learned to fuse results by optimizing the retrieval performance of the final list. The model further investigated the potential of applying deep neural networks to estimating the document feature function. We also developed a set of general query-list features that are effective and available for a general fusion task, compensating for the absence of specific reformulation features (e.g., query clarity and drift) used in λ-Merge.

We evaluated our model using two TREC FedWeb test sets for collection fusion, and two TREC Web Track test sets for data fusion. The results demonstrated that incorporating the vertical signals consistently improves the performance using only document and list feature functions. The results further showed that the LTM-Deep approach can significantly outperform standard baselines and state-of-art techniques.

Developing an effective optimization framework for result merging is a complex problem. While our experiments have shown promising results, it will be important for us to consider different possibilities in infrastructure and feature design as discussed in Section 5.4. Since the effectiveness of deep structures depends on the quality and quantity of training data, using larger test sets such as query search logs will be important, and we expect the performance can be further improved.

Acknowledgements

This work was supported in part by the Center for Intelligent Information Retrieval and in part by NSF IIS-1160894. 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.

7. REFERENCES

[1] J. A. Aslam and M. Montague. Bayes optimal metasearch: A probabilistic model for combining the results of multiple retrieval systems (poster session). In *Proceedings of SIGIR*, SIGIR '00, pages 379–381, 2000.

[2] J. A. Aslam and M. Montague. Models for metasearch. In *Proceedings of SIGIR*, SIGIR '01, pages 276–284, 2001.

[3] S. M. Beitzel, E. C. Jensen, A. Chowdhury, D. Grossman, O. Frieder, and N. Goharian. On fusion of effective retrieval strategies in the same information retrieval system. *JASIST*, 55:859–868, 2004.

[4] Y. Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, Jan. 2009.

[5] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, pages 437–478. Springer, 2012.

[6] C. Burges, R. Ragno, and Q. Le. Learning to rank with non-smooth cost functions. In *Advances in Neural Information Processing Systems 19*, January 2007.

[7] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proceedings of ICML*, ICML '05, pages 89–96, 2005.

[8] C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. Technical report, June 2010.

[9] J. Callan. Distributed information retrieval. In *In: Advances in Information Retrieval*, pages 127–150. Kluwer Academic Publishers, 2000.

[10] J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *Proceedings of SIGIR*, SIGIR '95, pages 21–28, 1995.

[11] G. V. Cormack, C. L. A. Clarke, and S. Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of SIGIR*, SIGIR '09, pages 758–759, 2009.

[12] W. B. Croft. Combining approaches to information retrieval. *Advances in Information Retrieval, Chapter 1, The Information Retrieval Series*, 7:1–36, 2000.

[13] T. Demeester, D. Trieschnigg, D. Nguyen, and D. Hiemstra. Overview of the trec 2013 federated web search track. In *Proceedings of TREC*, 2013.

[14] T. Demeester, D. Trieschnigg, D. Nguyen, Z. K., and D. Hiemstra. Overview of the trec 2014 federated web search track. In *Proceedings of TREC*, 2014.

[15] L. Deng, X. He, and J. Gao. Deep stacking networks for information retrieval. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3153–3157, May 2013.

[16] L. Deng and D. Yu. Deep learning: Methods and applications. Technical Report MSR-TR-2014-21, January 2014.

[17] E. A. Fox and J. A. Shaw. Combination of multiple searches. In *The Second Text REtrieval Conference (TREC-2)*, pages 243–252, 1994.

[18] F. Guan, S. Zhang, C. Liu, X. Yu, Y. Liu, and X. Cheng. Ictnet at federated web search track 2014. In *Proceedings of TREC*, 2014.

[19] D. Hawking and P. Thomas. Server selection methods in hybrid portal search. In *Proceedings of SIGIR*, SIGIR '05, pages 75–82, 2005.

[20] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of CIKM*, CIKM '13, pages 2333–2338, 2013.

[21] S. Jin and M. Lan. Simple may be best - a simple and effective method for federated web search via search engine impact factor estimation. In *Proceedings of TREC*, 2014.

[22] A. Khudyak Kozorovitsky and O. Kurland. Cluster-based fusion of retrieved lists. In *Proceedings of SIGIR*, SIGIR '11, pages 893–902, 2011.

[23] O. Kurland, A. Shtok, D. Carmel, and S. Hummel. A unified framework for post-retrieval query-performance prediction. In *Proceedings of ICTIR*, ICTIR'11, pages 15–26, 2011.

[24] J. H. Lee. Analyses of multiple evidence combination. In *Proceedings of SIGIR*, SIGIR '97, pages 267–276, 1997.

[25] D. Lillis, F. Toolan, R. Collier, and J. Dunnion. Probabilistic data fusion on a large document collection. *Artif. Intell. Rev.*, 26(1-2):23–34, Oct. 2006.

[26] D. Lillis, F. Toolan, R. Collier, and J. Dunnion. Extending probabilistic data fusion using sliding windows. In *Proceedings of ECIR*, ECIR'08, pages 358–369, 2008.

[27] G. Mesnil, X. He, L. Deng, and Y. Bengio. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *INTERSPEECH*, pages 3771–3775, 2013.

[28] D. Metzler and W. B. Croft. A markov random field model for term dependencies. In *Proceedings of SIGIR*, SIGIR '05, pages 472–479, 2005.

[29] T. Mikolov, W.-t. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751. Citeseer, 2013.

[30] M. Montague and J. A. Aslam. Condorcet fusion for improved retrieval. In *Proceedings of CIKM*, CIKM '02, pages 538–548, 2002.

[31] A. Mourao, F. Martins, and J. Magalhaes. Novasearch at trec 2013 federated web search track: Experiments with rank fusion. In *The 22nd Text Retrieval Conference*, TREC 13, 2013.

[32] D. Sheldon, M. Shokouhi, M. Szummer, and N. Craswell. Lambdamerge: Merging the results of query reformulations. In *Proceedings of WSDM*, WSDM '11, pages 795–804, 2011.

[33] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of CIKM*, CIKM '14, pages 101–110, 2014.

[34] M. Shokouhi. Segmentation of search engine results for effective data-fusion. In *Proceedings of ECIR*, ECIR'07, pages 185–197, 2007.

[35] M. Shokouhi and L. Si. Federated search. *Found. Trends Inf. Retr.*, 5(1):1–102, Jan. 2011.

[36] M. Shokouhi and J. Zobel. Robust result merging using sample-based score estimates. *ACM Trans. Inf. Syst.*, 27(3):14:1–14:29, May 2009.

[37] A. Shtok, O. Kurland, D. Carmel, F. Raiber, and G. Markovits. Predicting query performance by query-drift estimation. *ACM Trans. Inf. Syst.*, 30(2):11:1–11:35, May 2012.

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

[39] C. C. Vogt and G. W. Cottrell. Fusion via a linear combination of scores. *Inf. Retr.*, 1(3):151–173, Oct. 1999.

[40] E. M. Voorhees, N. K. Gupta, and B. Johnson-laird. The collection fusion problem. In *Proceedings of the TREC*, pages 95–104, 1995.

[41] Q. Wu, C. J. Burges, K. M. Svore, and J. Gao. Adapting boosting for information retrieval measures. *Inf. Retr.*, 13(3):254–270, June 2010.

[42] S. Wu and S. McClean. Performance prediction of data fusion for information retrieval. *Inf. Process. Manage.*, 42(4):899–915, July 2006.