Learning to Aggregate Vertical Results into Web Search Results

Jaime Arguello[*]
School of Information and Library Science
University of North Carolina
Chapel Hill, NC, USA
jarguello@unc.edu

Fernando Diaz
Yahoo! Labs New York
New York, NY, USA
diazf@yahoo-inc.com

Jamie Callan
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA, USA
callan@cs.cmu.edu

ABSTRACT

Aggregated search is the task of integrating results from potentially multiple specialized search services, or *verticals*, into the Web search results. The task requires predicting not only which verticals to present (the focus of most prior research), but also predicting where in the Web results to present them (i.e., above or below the Web results, or somewhere in between). Learning models to aggregate results from multiple verticals is associated with two major challenges. First, because verticals retrieve different types of results and address different search tasks, results from different verticals are associated with different types of predictive evidence (or features). Second, even when a feature is common across verticals, its predictiveness may be vertical-specific. Therefore, approaches to aggregating vertical results require handling an inconsistent feature representation across verticals, and, potentially, a vertical-specific relationship between features and relevance. We present 3 general approaches that address these challenges in different ways and compare their results across a set of 13 verticals and 1070 queries. We show that the best approaches are those that allow the learning algorithm to learn a vertical-specific relationship between features and relevance.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Information Storage and Retrieval

General Terms

Algorithms

General Terms

aggregated search, federated search, query intent, learning to rank

[*]Work done at Carnegie Mellon University.

1. INTRODUCTION

In addition to providing Web search, commercial search engines provide access to specialized search services (referred to as *verticals*) that focus on a specific information-seeking task (e.g., search for news, images, video, local businesses, items for sale, weather forecasts, etc.). Aggregated search refers to the detection and integration of relevant vertical content into the Web search results page. The problem is typically decomposed into two sub-tasks. It is impractical, if not impossible, to issue the query to every vertical. Thus, the first sub-task (*vertical selection*) is to predict *which* verticals, if any, are relevant [2, 3, 6]. Typically, this is done without issuing the query to the vertical [2, 3]. The second sub-task (*vertical results presentation*) is to predict *where* in the web results to display those verticals selected [16, 1]. This work focuses on the vertical results presentation task.

If we assume that vertical results must be presented in specific positions relative to the Web search results (i.e., above, below, or in between certain Web results), then the vertical results presentation task can be cast as *block-ranking.* A *block* is defined as a short sequence of Web or same-vertical results which must be presented grouped together—vertically (e.g., *news*) or horizontally (e.g, *images*)—in the final search results page. A Web block corresponds to a set of Web results which cannot be partitioned in the final presentation (e.g., Web results 1-3, 4-6, and 7-10). A vertical block corresponds to the top results from a particular vertical. The goal of block-ranking is to predict a ranking of blocks that approximates, based on some metric, a gold standard ranking. We propose and evaluate three supervised machine learning approaches to block-ranking.

Casting block-ranking as a supervised machine learning problem is associated with two main challenges. First, because verticals retrieve different types of results (e.g., news articles, images, videos, local business information, weather forecasts), blocks from different verticals are associated with different types of predictive evidence or features. For example, *news* results are associated with a publication date, *local* results are associated with a geographical location, and *community Q&A* results are associated with a number of suggested answers. Thus, block-ranking requires approaches that can handle an inconsistent feature representation across verticals. Secondly, even if a feature is common to multiple verticals, it may not be equally predictive. For example, the number of results retrieved by the vertical will likely be more predictive for *news* than *weather*, which retrieves at most a *single* result. Alternatively, a feature may be predictive for different verticals, but in the *opposite* direction. For example, the query-term "pics" may be positive evidence for *images*, but negative evidence for *shopping*. Thus, block-ranking may require approaches that can exploit a vertical-*specific* predictive relationship between features and relevance. We propose methods that address both challenges in different ways.

We evaluate approaches that rank blocks as a function of a set of features. We partition features into two general classes: *pre-retrieval* and *post-retrieval* features. During vertical results presentation, we assume that the system has already issued the query to each vertical, or at least those predicted relevant during vertical selection. Thus, post-retrieval features can be derived directly from the vertical results. We investigate the cost-benefit of post-retrieval features. Are they useful for predicting *where* a vertical should be presented? Can they *also* be used to re-evaluate vertical selection decisions in light of the vertical results? Answering these questions affects, for example, whether the end-to-end system should attempt to issue the query to as many verticals as possible (to derive these features), or whether it should cache post-retrieval features for future impressions of the query.

2. RELATED WORK

Research in aggregated search can be described along several dimensions: number of verticals considered, number of positions in which vertical content can be integrated, and sources of training data. The earliest work in aggregated search integrated the content from a single vertical above the first Web result [6, 10, 12]. Unfortunately, evaluating the performance of a single vertical in isolation ignores possible contention with other verticals. As a result, more realistic models were developed which considered several verticals simultaneously [2, 3, 7]. Limiting integration to the top position is rather crude when considering the nuanced nature of user intent. For example, if a query's dominant intent is navigational, there may be a secondary, less popular, news intent which should be surfaced lower in the ranked list. In order to address this, more recent work has focused on integrating at arbitrary positions in the ranked list [1, 16]. Finally, all of the prior work in aggregated search uses machine learning methods and, as a result, sources of training data become important. Training data has included editorial judgments [2, 1, 12], click information [6, 10, 16], and indirect labels from other verticals [3].

We focus our experiments on methods of learning orderings of generic Web and vertical content using pairwise labels [14]. This is quite different from prior work which focuses on minimizing misclassification of vertical relevance [2, 3, 7, 12] or regressing against a click-based target [6, 10, 16]. In the language of 'learning to rank', these previous approaches are based on pointwise sources of relevance while our approach focuses on pairwise sources.

3. PROBLEM DEFINITION

At query time, the aggregated search system issues the query to the Web search engine and to those verticals selected, which we refer to as the *candidate* verticals. The aggregation task is subject to a set of layout constraints. We assume the following constraints. First, we assume that results from the same vertical must be presented together. Second, vertical results can only be embedded in specific positions relative to the Web results. We assume four slotting positions: above the first Web result, between Web results 3 and 4, between results 6 and 7, and below the tenth Web result. Third, Web results are always presented and maintain their original order. Fourth, each candidate vertical is associated with a set of top results, which are given. That is, the system does not predict which results from a particular vertical to present. Finally, we assume that users prefer to not see non-relevant vertical results even below the last Web result. The system is free to suppress a candidate vertical, for example, based on its results.

Combined, these layout constraints form a set of blocks. Each candidate vertical forms its own vertical block and our four slotting positions divide the top 10 Web results into three Web blocks: Web results 1-3 (w_1), 4-6 (w_2), and 7-10 (w_3). We define the block-ranking task as follows. Let $\mathcal{B}_q$ denote the set of Web and vertical blocks associated with query q and let σ_q^* denote the optimal ordering of $\mathcal{B}_q$ for query q, referred to as the *reference* ranking. The objective of block ranking is to predict a ranking of blocks σ_q that approximates σ_q^*. The quality of the approximation can be measured using a rank-based distance metric, such as Kendall's τ. We discuss this in Section 6.5.

In addition to deciding where to present vertical results, the task is also to filter non-relevant candidate verticals. Suppressed verticals are modeled using an imaginary "end of search results" block, denoted as *eos*. The *eos* block is included in $\mathcal{B}_q$, and, therefore, appears in both σ_q and σ_q^*. Let $\sigma_q(i)$ denote the rank of block i in σ_q. Blocks ranked above *eos* are considered to be displayed to the user and those ranked below it are suppressed. Because the sub-ranking of blocks below rank $\sigma_q(eos)$ is not observed by the user, for the purpose of comparing σ_q with σ_q^*, all blocks ranked below *eos* in σ_q are considered tied at rank $\sigma_q(eos) + 1$.

4. FEATURES

We propose machine learning approaches to rank blocks as a function of a set of features. We use various types of features which we believe are predictive of a particular block's relevance to a query. These can be divided into two general classes. *Pre-retrieval features* can be generated without issuing the query to the Web or vertical search engine. These include, for example, the topic of the query or whether the query contains a particular named-entity type (e.g., the name of a person, product, or location). *Post-retrieval features* must be generated *after* the query is issued to the Web or vertical search engine. These include, for example, the total number of results retrieved by the block's search engine or the average text-similarity between the query and the results presented in the block. We investigate the contribution of post-retrieval features to performance.

4.1 Pre-retrieval Features

Pre-retrieval features can be generated before the query is issued to the Web or vertical search engine.

Named-Entity Type Features.

These binary features correspond to named-entity types possibly appearing in the query. Queries were automatically annotated using the BBN IdentiFinder named-entity tagger [4]. Named-entity features include *location* (possibly predictive for *local*, *maps*, and *weather*), *product* (possibly predictive for *shopping*), *person* (possibly predictive *news* and *images*), and *organization* (possibly predictive for *finance*). In total, we focused on 24 named-entity types. Each binary feature equals 1 if the named-entity type appears at least once in the query and 0 otherwise.

Category Features.

Some verticals may be topically focused. Thus, knowing the general topic of the query may help in block ranking. We focused on 30 topical categories, derived as follows. First, we selected 150 categories from the Open Directory Project (ODP) hierarchy and crawled Web documents associated with these ODP nodes. Then, in order to reduce the number of category features, these 150 categories were clustered into 30 clusters. Clustering was done using complete-link agglomerative clustering [15]. The distance between ODP categories was computed using the symmetric Kullback-Leibler divergence [8] between category language models. We used unigram language models with add-one smoothing to avoid zero probabilities. Let θ_i denote the language model associated with cluster/category i. We set the i^{th} category feature for query q according to, $\frac{1}{\mathcal{Z}} \prod_{w \in q} P(w|\theta_i)$ where $\mathcal{Z}$ normalizes across clusters/categories.

Click-through Features.

User clicks are often viewed as surrogates for relevance. The queries associated with clicks on a particular document convey the types of information needs the document satisfies. Likewise, the queries associated with clicks on vertical results convey the types of information needs the vertical satisfies. Our click-through features harness this type of evidence by considering the similarity between the query and those associated with clicks on vertical content (or content very similar to the vertical's).

Click-through data was derived from the AOL query-log. We derived one click-through feature per vertical as follows. First, for each vertical, we manually selected a set of Web domains which we believe have content closely related to the vertical. For *local*, we selected www.local.yahoo.com, www.citysearch.com, and www.yellowpages.com. Then, we constructed vertical-specific (unigram) language models using all queries (allowing duplicates) associated with click-events on the vertical's corresponding domains. Finally, given a query, we generate one feature per vertical based on the query generation probability given the vertical's query-based language model. Given query q, we set the click-through feature for vertical i according to, $\frac{1}{\mathcal{Z}} \prod_{w \in q} P(w|\theta_i)$ where $\mathcal{Z}$ normalizes across verticals.

Vertical-Intent Features.

Users often express vertical-intent *explicitly* using keywords such as "news" (for the *news* vertical), "pics" (for the *images* vertical) or "buy" (for the *shopping* vertical). The goal of our explicit vertical-intent features is to determine vertical relevance based on how often the query co-occurs with keywords used in explicit requests for the vertical. For example, given the query "britney spears", we may predict that *images* is relevant because users often issue the query "britney spears pics". Co-occurrence statistics were derived from the AOL query-log.

Vertical-intent features were generated as follows. First, we manually associated each vertical with a small set of keywords which we believe are often used in explicit requests for the vertical. For example, the *images* vertical was associated with "picture(s)", "photo(s)", "pic(s)", and "image(s)".[1] Then, to measure the affinity between the query and a particular vertical, we use the chi-squared statistic to measure the lack of independence between two events: the occurrence of the query in the AOL query-log and the occurrence of any of the vertical's vertical-intent keywords. To reduce sparsity (particularly for long queries), we compute the chi-squared statistic for each query-term individually and use the geometric average. The geometric average (rather than the arithmetic average) was used to favor queries with terms that *consistently* co-occur with keywords used in explicit requests for the vertical.

4.2 Post-retrieval Features

Post-retrieval features must be generated after the query is issued to the Web or vertical search engine.

Hit Count Features.

This feature considers the number of results retrieved from the Web or vertical search engine. For some verticals, an abundance of query-related content in the index may be predictive of its relevance. This may be true, for example, for *news*, where the rate of content production may correlate with the rate of content demand. However, we do not expect this feature to be useful for every vertical. For example, the number of retrieved results contributes no useful information for verticals that retrieve at most a *single* result (*finance*, *maps*, and *weather*).

Temporal Features.

Some verticals may be time sensitive. Prior work shows that recency is important in news search [6]. Temporal information was available for four verticals: *news*, *blogs*, *community Q&A*, and *twitter*. Our assumption is that, for these verticals, users care primarily about recent results. If this is true, then the average age of results presented in the block should affect its relevance. Temporal features are generated as follows. For each individual vertical result, we measure the elapsed time (in hours) between the current and created date/time. We included four features for each time-sensitive vertical: the minimum, maximum, mean, and standard deviation of the elapsed time across results within the block.

Text-Similarity Features.

The goal of these features is to characterize the text-similarity between the query and the results presented in the block. The challenge in deriving text-similarity features is that results from different sources (i.e., results from the Web search engine and from different verticals) are associated with *different* sets of textual representations. For example, each Web result is associated with three representations: a title, URL, and summary snippet. Each *community Q&A* result is associated with two representations: a question and an optional "best answer". Each *weather* result is associated with a single representation: the location, which we define as the concatenation of the city, state, and country of the weather forecast.

[1]The full vertical-to-keyword and vertical-to-domain mappings are available at `http://www.ils.unc.edu/~jarguell/cikm11/`.

Text-similarity features are generated for a query-block pair in two steps. In the first step, for each result within the block, we measure the text similarity between the query and each representation associated with the result. We use four different text-similarity measures: (1) the cosine similarity between the query and the representation, (2) the maximum number of query-terms appearing consecutively in the representation, (3) the percentage of query-terms appearing in the representation, and (4) the percentage of the representation corresponding to a query-term. Similar text-similarity features were used in prior learning-to-rank research (for document ranking) [14, 9].

The second step depends on whether the block-type is associated with a single result per block (e.g., *weather*, *finance*, and *maps*) or multiple results per block (e.g., *news*, *local*, and *shopping*). For block-types with a single result, we simply include our four similarity measures for each of its text representations. For block-types with multiple results, for each query-representation similarity measure, we use the minimum, maximum, mean, and standard deviation across results within the block.

4.3 Summary of Features

Pre-retrieval features are independent of the block. Thus, every block-type is associated with the same set of 80 pre-retrieval features: 24 named-entity type features, 30 category features, 13 click-through features, and 13 vertical intent features. Notice that we included all 13 click-through features (one per vertical) and all 13 vertical intent features (one per vertical) in every block's feature representation, irrespective of its type. Post-retrieval features, as opposed to pre-retrieval features, are derived directly the block (e.g., the average text-similarity between the query and the title of results within the block) or from the search engine's response to the query (e.g., its hit count). Different block-types were associated with a different set of post-retrieval features. Hit count features were omitted for verticals that retrieve at most a *single* result (i.e., *finance*, *maps*, and *weather*). Temporal features were only available for *news*, *blogs*, *community Q&A*, and *twitter*. Text-similarity features are inconsistent because different block-types are associated with different representations and a different number of results—block-types with multiple results per block require aggregating evidence by taking the minimum, maximum, mean, and standard deviation of query-representation similarities across results.

5. BLOCK-RANKING APPROACHES

As previously mentioned, casting block-ranking as a supervised machine learning problem is associated with two main challenges. First, different types of blocks are associated with different features. Second, even when a feature is common to multiple block-types, it may have a type-*specific* relationship with relevance. We propose three general approaches which address both challenges in different ways.

5.1 Classification Approach

Our classification framework takes the form of n independent binary classifiers (one per vertical). We choose to use logistic regression due to its prediction accuracy and training speed on large-scale classification tasks [13].

Each binary classifier is trained to predict whether a particular vertical should be presented (ranked above *eos*) or suppressed (ranked below *eos*). While training the classifier for vertical v, a query is considered a positive instance if v is ranked above *eos* in the reference ranking σ_q^* and a negative instance otherwise. To compensate for class imbalance (verticals are more often suppressed), each positive training instance is weighted according to the number of negative instances in the training set and vice-versa [5].

The classification approach produces a block-ranking by assigning vertical blocks to slots. Consistent with our layout constraints, we assume four vertical slotting positions relative to the Web results: slot s_1 (above w_1), slot s_2 (between w_1 and w_2), slot s_3 (between w_2 and w_3), and slot s_4 (between w_3 and *eos*). In the output ranking, a slot may contain zero or more vertical blocks.

The classification approach combines all n vertical-specific binary classifiers as follows. First, the query, represented as a vector of features, is submitted to each candidate vertical's classifier. Each classifier outputs a prediction probability that its vertical should be presented (i.e., ranked above *eos* in the predicted ranking σ_q). Let $P(\sigma_q(v) < \sigma_q(eos))$ denote the prediction probability that v should be presented. Then, each candidate vertical is assigned to a slot (or is suppressed) using four threshold parameters τ_{1-4}. Vertical v is assigned to slot x if $P(\sigma_q(v) < \sigma_q(eos)) \geq \tau_y \ \forall \ x \leq y$. In other words, vertical v is assigned to the highest slot for which v's prediction probability is greater than or equal to all thresholds below it. At this point, vertical blocks assigned to the same slot are tied. Finally, ties are broken by ordering vertical blocks within the same slot by descending order of prediction probability.

The classification approach addresses the two challenges mentioned above by training a different binary classifier per vertical. Each classifier adopts its own feature representation, which is unique to the vertical, and learns a vertical-specific relationship between features and block relevance.

5.2 Voting Approach

In the classification approach, each independent binary classifier is trained to predict whether a particular vertical should be presented or suppressed. Our voting approach also combines independent binary classifiers. However, these are trained to make more fine-grained predictions. Independent binary classifiers are trained to predict the relative ordering between pairs of blocks of a particular type. Each classifier is trained to predict whether block i (of a particular type) should be ranked above or below block j (of another particular type) for a given query. The voting approach uses one binary classifier per block-type pair.

The training phase proceeds as follows. Recall that $\mathcal{B}_q$ denotes the set of block associated with query q and includes one block per candidate vertical, all three Web blocks (w_{1-3}), and the *eos* block. While training a classifier specific to a block-type pair, the query is considered a positive or negative instance depending on the pair's relative rank. Certain block-type pairs occur more frequently in one order versus the other. To correct for class imbalance, each positive training instance is weighted according to the number of negative instances in the training set and vice-versa [5].

To predict a block-ranking for query q, first, every block-pair $i, j \in \mathcal{B}_q$ is submitted to the appropriate classifier, depending on the type of i and the type of j. Let $P(\sigma_q(i) < \sigma_q(j))$ denote the prediction probability that i should be ranked above j. This probability can be treated as a preference score between i and j. The voting approach produces the output block-ranking σ_q by combining these preference scores as input to the Schulze voting method [17].

As in the classification approach, each binary classifier is associated with a unique feature representation. More specifically, each classifier is associated with three sets of features: one set of pre-retrieval features, which are independent of the block-types under consideration, and two separate sets of post-retrieval features (one set specific to each type in the block-type-pair).

The voting approach addresses both challenges mentioned above (block-type-specific features and a potentially different predictive relationship across types) by training a different binary classifier per block-type pair. Each classifier adopts its own feature representation and learns a predictive relationship that is specific to the block-type-pair.

Given n verticals and m non-vertical block-types, the total number of binary classifiers used by the voting approach is given by $\binom{n+m}{2} - \binom{m}{2}$. The second term accounts for the fact that Web results are always presented and always ranked in their original order (i.e., $\sigma_q(w_1) < \sigma_q(w_2) < \sigma_q(w_3) < \sigma_q(eos)$). Thus, it is not required to learn a classifier to determine the relative ordering between pairs of non-vertical blocks. In our case, $n = 13$ and $m = 4$, which results in 130 independent binary classifiers. The large number of binary classifiers used by this method may be viewed as a disadvantage. An alternative to learning one model per block-type pair is to learn one model per vertical/non-vertical block-type pair. This results in four models per vertical: three which predict the vertical's relevance compared to each Web block (w_{1-3}) and one which predicts its relevance compared to *eos* (i.e., it predicts whether to display/suppress the vertical). As before, given a query, every block-pair $i, j \in \mathcal{B}_q$, where one is a vertical- and the other a non-vertical block, is submitted to the appropriate classifier. Then, the output prediction probabilities are combined as the input to the Schulze voting method in order to derive σ_q.

5.3 Learning to Rank Approaches

The block-ranking task can also be cast as a learning to rank (LTR) problem. Many different learning to rank methods have been proposed. In this work, we adopt a *pairwise* learning to rank approach. Pairwise approaches optimize over the set of pairwise preferences implicit in the training data. More specifically, we adopt a method that solves the classic RankSVM optimization problem, first proposed by Joachims [9]. RankSVM learns a linear model $f_{\mathbf{w}}$ parameterized by feature weight vector $\mathbf{w}$.

Casting block-ranking as an LTR problem requires training a *single* model $f_{\mathbf{w}}$ to predict a block's rank irrespective of its type. In our situation, this is problematic because different block-types are associated with different features (i.e., some features may be specific to a handful of types and some may be unique to a particular one). In addition, it is problematic because those features that are common to *multiple* types (e.g., whether the query contains a city name) may be predictive for some types more than others, or even predictive for different types in the opposite direction. Next, we propose three LTR variants which address these challenges in different ways. Each variant makes a different assumption about how features may be correlated with block relevance across block-types.

Equally Correlated Features.

One alternative is to assume that each feature is equally predictive of block relevance (in the same direction) independent of the block-type. The feature representation is as follows. Pre-retrieval features are independent of the block. This model uses a *single* copy of each pre-retrieval feature. Post-retrieval features are block-specific (i.e., they are generated directly from the block or the block's search engine results). Similar to pre-retrieval features, this approach *also* uses a *single* copy of each post-retrieval feature. If a block is not associated with a particular post-retrieval feature, then the feature is zeroed-out in that instance. Consider, for example, our post-retrieval features which determine the text-similarity between the query and the summary snippets presented in the block. These features are only associated with *news* and Web blocks w_{1-3}. Therefore, if the block is not one of these types, all these features are zeroed-out.

This approach assumes that features are equally correlated with relevance irrespective of the block-type. Once trained, model $f_{\mathbf{w}}$ will apply the *same* weight to a feature independent of the instance's block-type. We denote this LTR variant as `LTR-G` because it assumes a vertical-*general* relationship between features and relevance.

Uniquely Correlated Features.

This approach makes the opposite assumption as the previous one. It assumes that every feature—whether it is a pre- or post-retrieval feature—is uniquely correlated with relevance across different block-types. The feature representation is as follows. We make a separate, block-type-specific copy of each feature. So, for example, given 17 block-types (13 verticals + 3 Web blocks and the *eos* block), we make 17 copies of each pre-retrieval feature (one per block-type). Given an instance, all copies are zeroed-out except for those corresponding to the instance's block-type. For post-retrieval features, we make one copy per block-type for which the feature is available. Consider, for example, our temporal features, which are available for blocks from *blogs*, *community Q&A*, *news*, and *twitter*. We make 4 copies of each temporal feature.

This approach assumes that features are correlated differently with relevance depending on the block-type. Once trained, model $f_{\mathbf{w}}$ can apply a *different* weight to a feature, depending on the instance's block-type. While this added flexibility may be advantageous, the increased number of features may introduce predictive noise and result in overfitting. Thus, this LTR variant may require more training data than `LTR-G`. We denote this LTR variant as `LTR-S` because it assumes a vertical-*specific* relationship between features and relevance.

Equally and Uniquely Correlated Features.

The previous two approaches make opposite assumptions: features are either equally correlated or uniquely correlated with relevance for different block-types. A third alternative is to make neither assumption *a priori*, but to give the algorithm the freedom to exploit both types of relationships using training data.

For this approach, we maintain a single copy of each pre- and post-retrieval feature which is shared across all block-types. As before, if an instance's block-type is not associated with a shared feature, the feature is zeroed-out for that instance. In addition to these shared features, we make one block-type-specific copy of each pre- and post-retrieval feature. Given an instance, all copies corresponding to types other than the instance's block-type are zeroed-out. The canonical feature representation for this approach is the union of features used by the previous two approaches.

This approach makes no assumption about how a feature is correlated with relevance across block-types. If a feature is equally correlated across block-types, then the algorithm can assign a large (positive or negative) weight to the copy of the feature which is shared across types. Alternatively, if a feature is correlated differently for different block-types, then the algorithm can assign a large positive weight to some copies of the feature and a large negative weight with to others. We denote this LTR variant as `LTR-GS` because it has the flexibility of learning a vertical-specific and vertical-general relationship for each feature. Of all three LTR variants, `LTR-GS` has the largest number of features and may therefore need the most training data to avoid overfitting.

6. METHODS AND MATERIALS

We evaluate performance using a set of 13 verticals and 1070 queries. Our evaluation methodology is based on that proposed in Arguello *et al.* [1]. For each query, a reference block-ranking σ_q^* is derived from human preference judgements on block-pairs $i, j \in \mathcal{B}_q$. These 1070 reference presentations are used for training and evaluation. That is, during training, algorithms are tuned to predict a block-ranking σ_q that approximates the reference block-ranking σ_q^*. During testing, we evaluate algorithms based on the quality of their approximation on unseen queries. Following Arguello *et al.* [1], we evaluate the predicted block-ranking σ_q based on its *generalized* Kendall's τ distance to σ_q^*, denoted by $K^*(\sigma_q^*, \sigma_q)$ [11]. Arguello *et al.* [1] presented a user study which shows that when assessors strongly prefer one block-ranking over another, the preferred block-ranking is scored as being superior by this metric. That is, on average, the preferred presentation is the one closest to σ_q^* based on K^*.

Approaches are compared based on average performance across queries using 10-fold cross-validation. Unless otherwise stated, statistical significance is tested using a one-tailed paired t-test (at the $p < 0.05$ level) by comparing performance across test queries (i.e., the concatenation of all 10 test-folds). Next, we describe the methodology for deriving σ_q^*, the human relevance assessment process, the evaluation metric $K^*(\sigma_q^*, \sigma_q)$, our set of verticals and queries, and some algorithm implementation details.

6.1 Deriving the Reference Block-Ranking

We provide a general overview of how σ_q^* is derived from human relevance assessments and refer the reader to Arguello *et al.* [1] for more details.

Given query q, the reference presentation σ_q^* is derived from *redundant* pairwise preference judgements collected for *all* block-pairs $i, j \in \mathcal{B}_q$. That is, every candidate block-pair for the query is shown to multiple assessors, who are asked to state a preference or to state that both blocks i and j should be suppressed. Let π_q denote the full set of block-pair judgements collected for query q and let $\pi_q(i, j)$ denote the strength with which block i is preferred over j given q. We collected 3 redundant judgements per triplet (q, i, j) and set $\pi_q(i, j)$ equal to the number of assessors who prefer i over j given q. $\pi_q(eos, i)$ was set equal to the number of assessors who stated that i should be suppressed in conjunction with another block j. Finally, to derive σ_q^*, block-pair judgements π_q are input to the Schulze voting method [17], which converts these block-pair preferences into a ranking of blocks. A detailed explanation of the Schulze voting method is given in Schulze [17] and a description of how it is used to infer σ_q^* from π_q is given in Arguello *et al.* [1].

6.2 Human Relevance Assessment

Block-pair assessments for 1070 queries were collected using Amazon's Mechanical Turk (AMT). Assessors were compensated US$ 0.01 per block-pair assessment. For a given triplet (i, j, q), assessors were presented with the query q and blocks i and j presented side-by-side in random order. The only difference between our assessments and those collected in Arguello *et al.* [1] is that assessors were not given a topic description for the query. Instead, they were instructed to judge blocks based on their best guess of the hypothetical user's intent. To do quality control, 10% of all block-pair assessments were traps. In a trap assessment, the assessor is given a triplet (q, i, j), but either block i or j is taken from a query other than q. If the assessor selects the extraneous block as the preferred block, we consider it a *failed* trap. Assessors with more than 2 failed traps had all their judgements removed and re-submited to AMT.

6.3 Verticals and Queries

We focused on 13 verticals constructed using free APIs from eBay (*shopping*), Google (*blogs*, *books*, *weather*), Recipe Puppy (*recipes*), Yahoo! (*community Q&A*, *finance*, *images*, *local*, *maps*, *news*), Twitter (*twitter*), and YouTube (*video*). For the *local*, *maps*, and *weather* verticals, queries were mapped to a geographical location using Yahoo!'s GeoPlanet API. For queries with no target location explicitly mentioned (e.g., "home depot"), we assumed a particular location and instructed assessors to interpret such queries as a resident of that location. Blocks were constructed using the vertical's top results, assuming a maximum number of results (e.g., 5) per vertical block.

A set of 1,070 queries for model learning and evaluation was collected using sampling. We are interested not only in performance across queries, but also across verticals. Therefore, our sampling approach is biased towards coverage across our set of 13 verticals. For space reasons, we omit the details of our sampling approach.[2]

6.4 Modeling Bias Towards Vertical Results

In some cases, one may prefer a system that is biased towards vertical results. The aggregated web search provider may want to tune the system so that it more frequently presents vertical results and/or presents them higher in the output presentation. This may occur, for example, in order to gather user-interaction data or to satisfy contractual obligations with vertical providers or advertisers.

Vertical bias can be modeled in our evaluation framework using vertical *pseudo-votes*, a parameter p in the range $[0, \infty)$. As previously noted, $\pi_q(i, j)$ corresponds to the number of assessors who prefer i over j given q. A vertical bias can be introduced by incrementing this value by some number of pseudo-votes p, but only if i corresponds to a vertical block. If i and j correspond to blocks from different verticals, this method increments both $\pi_q(i, j)$ and $\pi_q(j, i)$ by

[2]A description of our sampling approach is available at `http://www.ils.unc.edu/~jarguell/cikm11`.

an equal number of pseudo-votes. This has the effect that, after producing σ_q^* using the Schulze voting method, vertical results are ranked higher in σ_q^*. However, the ranking of verticals relative to each other is unchanged. Modeling vertical bias using vertical pseudo-votes changes σ_q^* and, therefore, changes model learning and evaluation. Rather than select a single pseudo-vote parameter p, we learn and evaluate models across several values of p (i.e., $p = \{0, 1, 2, 3, 4, 5\}$). Across values of p, the verticals ranked higher in σ_q^* were *video*, *local*, *news*, *blogs*, and *community Q&A*.

6.5 Evaluation Metric

The primary evaluation metric is the *generalized* Kendall's τ distance between the predicted block-ranking σ_q and the reference block-ranking σ_q^*, denoted as $K^*(\sigma_q^*, \sigma_q)$ [11]. Generalized Kendall's τ is related to Kendall's τ, which measures the number of discordant pairs between two rankings of the same set of elements. For our purpose, however, Kendall's τ has a major limitation: it considers *all* discordant pairs equally important. In aggregated search, users typically scan results from top to bottom. Thus, discordant pairs at the top of the ranking should be more influential. As suggested in Kumar and Vassilvitskii [11], discordances at the top of the ranking can be made more influential by using a DCG-like cost function. See Arguello *et al.* [1] for details.

6.6 Implementation Details

The classification framework described in Section 5.1 requires tuning threshold parameters τ_{1-4}, which are used to slot verticals based on each classifier's prediction confidence value. In addition to these parameters, logistic regression requires tuning cost factor C, which determines the cost of misclassifying a training set instance. These 5 parameters were tuned using 10-fold cross-validation. More specifically, they were tuned for each train/test pair individually by doing a second level of 10-fold cross-validation on each primary fold's training set. An exhaustive search was used to find the parameter values with the best average performance across secondary train/test pairs. The voting approach described in Section 5.2 also uses logistic regression models: one per block-type-pair. The C parameter was tuned as described above. In both approaches, models were trained using the `LibLinear` toolkit.[3]

The prior probability that a particular vertical is ranked high is fairly low. This is a problem if during training the LTR model learns that the best alternative is to present w_{1-3} and suppress all verticals. In the classification and voting approach, we balanced the training data using instance weighting (i.e., assigning more weight to the minority class). Casting block-ranking as a learning-to-rank approach may also require some form of instance weighting.

Our approach to instance weighting is as follows. Let σ_q^w denote a block-ranking which presents Web blocks w_{1-3} in their original order and suppresses all verticals. Given a training query q, $-K^*(\sigma_q^*, \sigma_q^w)$, which is in the range $[-1, 1]$, measures the distance between σ_q^* and σ_q^w. If $-K^*(\sigma_q^*, \sigma_q^w)$ is close to $+1$, it means that σ_q^* has verticals presented in the top ranks. If $-K^*(\sigma_q^*, \sigma_q^w)$ is close to -1, it means that σ_q^* has Web blocks w_{1-3} presented in the top ranks and verticals presented in the low ranks or suppressed. Our approach to instance weighting is to replicate queries in the training set proportional to this distance. This has the effect that queries which deviate from σ_q^w are oversampled. The amount of replication is controlled using parameter α. First, we scale $-K^*(\sigma_q^*, \sigma_q^w)$ to zero min and unit max (using queries from the training set). Then, we multiply this value by α. Given training query q, the number of *additional* copies of q in the training set is given by $-\alpha K^*_{\text{min-max}}(\sigma_q^*, \sigma_q^w)$. Note that when $\alpha = 0$, no additional copies of q are added to the training set. We evaluate the effect of parameter α in Section 8.1.

In addition to parameter α, RankSVM has a regularization parameter λ. Both parameters were tuned using two levels of 10-fold cross-validation. An exhaustive search was used to find the parameter values with the best average performance across secondary train/test pairs. We trained LTR models using the `sofia-ml` toolkit.[4]

7. EVALUATION RESULTS

We evaluate three general approaches to block-ranking: the classification approach, the voting approach, and the LTR approach. One limitation of the voting approach is that it requires a large number of independent binary classifiers. Thus, we evaluate a second version of the voting approach that trains a binary classifier only for block-type-pairs where one type corresponds to a vertical block-type and the other corresponds to a non-vertical block-type. Both are included in this evaluation. We also include all three variants of the LTR approach, with parameter α tuned using cross-validation.

Results in terms of average $K^*(\sigma_q^*, \sigma_q)$ are presented in Table 1. As previously mentioned, pseudo-vote parameter p controls for vertical bias. The higher its value, the greater the bias towards verticals ranked high. Rather than choose a single parameter p, results are presented for $p = \{0, 1, 2, 3, 4, 5\}$. When $p = 0$, the σ_q^*'s used for training and evaluation have no vertical bias. The second row in Table 1 (`WEB`) corresponds to a degenerate baseline that suppresses all vertical results and simply presents w_{1-3} above *eos* in their original order.

Table 1 shows several meaningful trends. Performance for all methods decreases with greater values of p (this was expected for `WEB`). One possible reason is the following. As p increases, the number of queries with top-ranked verticals increases. This means that the room for error also increases. When p is low, ranking w_{1-3} above all verticals is a reliable and effective strategy.

When $p = 0$, `WEB` performs well. In fact, the only two approaches that significantly outperform `WEB` when $p = 0$ are `LTR-S` and `LTR-GS`. Significance with respect to `WEB` is not shown in Table 1. For values of $p \geq 3$, *all* block-ranking approaches significantly outperform `WEB`. Thus, given a vertical bias, any of the three general block-ranking approaches is better than presenting only Web results.

Both voting approaches perform similar to each other. This is interesting because the second version (which learns one binary classifier per vertical/non-vertical block-type) uses many fewer binary classifiers than the first (which learns one classifier per block-type). The performance for both voting approaches, however, deteriorates for $p \geq 2$.

The classification approach does surprisingly well considering that it uses a just one binary classifier per vertical. Its

[3]`http://www.csie.ntu.edu.tw/cjlin/liblinear/`

[4]`http://code.google.com/p/sofia-ml/`

Table 1: **Block-ranking results in terms of average $K^*(\sigma_q^*, \sigma_q)$. A $\vartriangle(\triangledown)$, $\blacktriangle(\blacktriangledown)$, and $\wedge(\vee)$ denotes significantly better(worse) performance compared to the classification, voting (all pairs), and LTR-G approaches, respectively.**

	$p = 0$	$p = 1$	$p = 2$	$p = 3$	$p = 4$	$p = 5$
WEB	$0.982^{\vartriangle}$	$0.959^{\vartriangle\vee}$	$0.896^{\triangledown\vee}$	$0.773^{\triangledown\blacktriangledown\vee}$	$0.656^{\triangledown\blacktriangledown\vee}$	$0.520^{\triangledown\blacktriangledown\vee}$
classification$^{\vartriangle\triangledown}$	$0.950^{\blacktriangledown\vee}$	$0.943^{\blacktriangledown\vee}$	$0.924^{\blacktriangle}$	$0.878^{\blacktriangle}$	$0.824^{\blacktriangle\wedge}$	$0.775^{\blacktriangle\wedge}$
voting (all pairs)$^{\blacktriangle\blacktriangledown}$	$0.984^{\vartriangle}$	$0.960^{\vartriangle\vee}$	$0.898^{\triangledown\vee}$	$0.830^{\triangledown\vee}$	$0.744^{\triangledown\vee}$	$0.734^{\triangledown\vee}$
voting (only v-w pairs)	$0.980^{\vartriangle\blacktriangledown}$	$0.956^{\vartriangle\blacktriangledown\vee}$	$0.898^{\triangledown\vee}$	$0.822^{\triangledown\vee}$	$0.760^{\triangledown\blacktriangle\vee}$	$0.734^{\triangledown}$
LTR-G $^{\wedge\vee}$	$0.980^{\vartriangle}$	$0.967^{\vartriangle\blacktriangle}$	$0.932^{\blacktriangle}$	$0.871^{\blacktriangle}$	$0.798^{\triangledown\blacktriangle}$	$0.754^{\triangledown\blacktriangle}$
LTR-S	$0.986^{\vartriangle\wedge}$	$0.970^{\vartriangle\blacktriangle}$	$0.937^{\vartriangle\blacktriangle}$	$0.883^{\blacktriangle\wedge}$	$0.838^{\vartriangle\blacktriangle\wedge}$	$0.792^{\vartriangle\blacktriangle\wedge}$
LTR-GS	$0.986^{\vartriangle\wedge}$	$0.973^{\vartriangle\blacktriangle\wedge}$	$0.936^{\vartriangle\blacktriangle}$	$0.882^{\blacktriangle\wedge}$	$0.843^{\vartriangle\blacktriangle\wedge}$	$0.795^{\vartriangle\blacktriangle\wedge}$

performance is competitive across all values of p, showing that it can be effectively trained to fit different degrees of vertical bias.

Comparing among the three LTR variants, performance is significantly worse for LTR-G. It never performs better and often performs significantly worse than LTR-S and LTR-GS, particularly for values of $p \geq 3$. For values of $p \geq 4$, LTR-G performs significantly worse than the classification approach. The improvement of LTR-S and LTR-GS over LTR-G reveals the importance of exploiting vertical-specific evidence. Imposing the constraint that features must be similarly correlated with relevance across different block-types degrades performance. LTR-G and LTR-S perform better by allowing the learning algorithm to exploit a block-type-specific relationship between features and block relevance.

Compared to the classification approach, LTR-S and LTR-GS both perform better. The improvement is significant for all values of p, except $p = 3$, where all three perform at the same level. We view this as a positive result in favor of casting block-ranking as a learning-to-rank task. In contrast with the classification approach, both of these LTR variants perform better and require training only a single model (rather than one per vertical). Relative to each other, LTR-S and LTR-GS are statistically indistinguishable across all values of p (statistical significance not shown in Table 1). We provide an explanation for this in Section 8.3.

8. DISCUSSION

8.1 Effect of Parameter α on LTR variants

The goal of parameter α is to focus LTR training on those queries that have verticals ranked high. Given enough evidence in favor of a particular vertical, we want to the LTR model to overcome the prior probability that the vertical is ranked low. Our method for instance weighting is to replicate queries in the training set proportional to the K^* distance between the reference block-ranking σ_q^* and a one that presents w_{1-3} and suppresses all verticals. Parameter α controls the amount of replication.

We were interested in the effect of parameter α on performance. Each LTR variant is evaluated under two conditions. In the first condition, $\alpha = 0$, which corresponds to *no replication*—each training query q appears just once in the training set. In the second condition, α is tuned by sweeping across $\alpha = \{0, 10, 25, 50\}$. Notice that $\alpha = 0$ (no replication) is a parameter choice. These results are identical to those in Table 1. Table 2 presents results (in terms of average $K(\sigma_q^*, \sigma_q)$) for all three LTR variants under these two conditions. As a second point of reference, we also present results for the classification approach. To conserve space, we limit results to $p = \{0, 2, 4, 5\}$.

Table 2: **Block-ranking results in terms of $K(\sigma_q^*, \sigma_q)$. A $\vartriangle(\triangledown)$ denotes significantly better(worse) performance compared to the classification approach. A $\blacktriangle(\blacktriangledown)$ denotes a significant improvement(drop) in performance for an LTR variant (with α tuned) compared to its counterpart for which $\alpha = 0$.**

	$p = 0$	$p = 2$	$p = 4$	$p = 5$
classification	0.950	0.924	0.824	0.775
LTR-G,α=0	$0.983^{\vartriangle}$	0.923	$0.775^{\triangledown}$	$0.701^{\triangledown}$
LTR-S,α=0	$0.986^{\vartriangle}$	0.928	$0.798^{\triangledown}$	$0.723^{\triangledown}$
LTR-GS,α=0	$0.986^{\vartriangle}$	$0.940^{\vartriangle}$	0.812	$0.743^{\triangledown}$
LTR-G	$0.980^{\vartriangle\blacktriangledown}$	$0.932^{\blacktriangle}$	$0.798^{\triangledown\blacktriangle}$	$0.754^{\triangledown\blacktriangle}$
LTR-S	$0.986^{\vartriangle}$	$0.937^{\vartriangle\blacktriangle}$	$0.838^{\vartriangle\blacktriangle}$	$0.792^{\vartriangle\blacktriangle}$
LTR-GS	$0.986^{\vartriangle}$	$0.936^{\vartriangle}$	$0.843^{\vartriangle\blacktriangle}$	$0.795^{\vartriangle\blacktriangle}$

The relative performance between LTR variants when $\alpha = 0$ (rows 3-5) is consistent with their relative performance when α is tuned (rows 6-8). That is, LTR-S and LTR-GS outperform LTR-G. Thus, we focus the discussion on LTR-S and LTR-GS. For both approaches, tuning α (vs. setting it to zero) has either no effect or significantly improves performance. For $p \geq 4$, setting $\alpha = 0$ degrades performance even compared to the classification approach. Thus, casting block-ranking as a learning-to-rank task benefits not only from allowing the algorithm to learn a block-type-specific relationship between features and relevance, it also benefits re-weighting training-phase instances in order to overcome the prior probability that verticals are ranked low.

8.2 Feature Contribution to Performance

A feature ablation study was conducted to test each feature group's contribution to overall performance, measured in terms of average $K(\sigma_q^*, \sigma_q)$. The analysis is conducted for both the classification approach and the LTR-S approach because they performed the best in Section 7. Each feature group was individually omitted and this model was compared to a model with access to all features. Because features may be correlated, a non-significant drop in performance does not necessarily mean that the feature group contributes no useful information.

Results are presented in Table 3 for the classification approach and the LTR-S approach. The top four feature groups are pre-retrieval features. The next four features are post-retrieval features. The last row corresponds to a model that ignores *all* post-retrieval features.

Table 3 shows several interesting results. The LTR-S approach appears to be slightly more robust to missing features. One possible reason is the following. The classification approach trains one independent binary classifier per vertical. The LTR-S approach, on the other hand, trains a single model. It may be that training a single model allows the LTR-S approach to better shift its focus towards block-types for which it is more confident.

Table 3: Feature ablation results for the classification and LTR-S approaches. A ▲(▼) denotes a significant improvement(drop) in performance compared to the model with access to all features.

classification approach

	$p = 0$	$p = 2$	$p = 4$	$p = 5$
all	0.950	0.924	0.824	0.775
no ne-type	0.003	-0.005	0.001	0.005
no cat	0.004	-0.83%▼	0.003	0.000
no click	-0.004	-1.71%▼	-1.97%▼	-2.05%▼
no vert-intent	-0.001	-1.15%▼	0.000	-0.009
no hit	0.003	-0.005	0.002	0.013
no loc	-0.001	-0.003	0.002	0.001
no temp	-0.002	0.000	-0.002	0.000
no text-sim	-0.003	-0.94%▼	-2.34%▼	-3.97%▼
no post-ret	-0.001	-0.86%▼	-2.29%▼	-3.44%▼

LTR-S

	$p = 0$	$p = 2$	$p = 4$	$p = 5$
all	0.986	0.937	0.838	0.792
no ne-type	-0.02%	0.19%	-0.42%	1.28%
no cat	-0.10%	0.31%	-0.35%	0.21%
no click	-0.04%	0.26%	-0.20%	-0.13%
no vert-intent	-0.03%	0.12%	0.16%	0.60%
no hit	-0.15%	-0.26%	-0.45%	-0.23%
no loc	-0.01%	0.12%	-0.65%	1.61%▲
no temp	0.01%	0.05%	-0.10%	0.65%
no text-sim	-0.09%	-0.41%	-2.60%▼	-3.44%▼
no post-ret	-0.02%	0.08%	-2.95%▼	-3.79%▼

Table 4: Feature contribution to per-vertical performance, based on AP. Statistical significance is tested using a one-tailed paired t-test, comparing across cross-validation test-folds. A ▲(▼) denotes a significant improvement(drop) in performance compared to the model with access to all features.

	q & a	blogs	book	finance	image
all features	0.284	0.431	0.088	0.638	0.323
no ne-type	10.15%▲	2.65%▲	34.41%▲	-18.69%▼	9.19%▲
no cat	15.49%▲	-6.61%▼	7.77%	21.95%▲	6.61%▲
no click	2.32%▲	0.32%	-1.29%	-14.35%▼	-9.23%▼
no vert-intent	7.80%▲	2.60%▲	1.97%	0.00%	-19.34%▼
no hit	5.53%▲	-0.61%▼	3.65%▲	2.44%▲	0.81%
no loc	0.00%	0.00%	0.00%	0.00%	0.00%
no temp	0.12%	-0.43%▼	0.00%	0.00%	0.00%
no text-sim	-8.44%▼	-12.35%▼	-31.16%▼	-3.38%▼	2.47%
no post-ret	-6.03%▼	-14.20%▼	-31.07%▼	-3.71%▼	2.20%
	local	map	news	recipe	shopping
all features	0.546	0.419	0.594	0.48	0.377
no ne-type	-3.97%▼	-1.99%	13.04%▲	6.97%▲	1.35%
no cat	-8.67%▼	-0.36%	21.92%▲	18.10%▲	-32.93%▼
no click	-10.44%▼	-16.86%▼	3.27%▲	-0.01%	-8.88%▼
no vert-intent	7.37%▲	-74.42%▼	2.73%	-3.63%▼	-0.65%
no hit	-2.89%▼	-1.94%▼	-4.21%▼	3.78%▲	0.21%
no loc	-8.84%▼	13.12%▲	0.00%	0.00%	0.00%
no temp	0.00%	0.00%	0.51%	0.00%	0.00%
no text-sim	-16.93%▼	0.00%	-48.88%▼	0.00%	-8.02%▼
no post-ret	-18.04%▼	13.15%▲	-45.26%▼	3.78%▲	-10.96%▼
	twitter	video	weather		
all features	0.053	0.571	0.938		
no ne-type	-14.19%▼	2.14%▲	4.55%▲		
no cat	-0.91%	3.35%▲	-17.31%▼		
no click	1.99%	-4.48%▼	-50.11%▼		
no vert-intent	0.31%	4.41%▲	-1.11%▼		
no hit	-5.90%▼	-0.75%▼	-0.45%		
no loc	0.00%	0.00%	-17.68%▼		
no temp	-5.71%▼	0.00%	0.00%		
no text-sim	-26.78%▼	-12.11%▼	0.00%		
no post-ret	-27.64%▼	-12.46%▼	-17.81%▼		

The features with the greatest drop in performance (at least for $p \geq 4$) are text-similarity features, which are a type of post-retrieval feature. This shows the importance of deriving evidence directly from those results presented in the block. Also, it suggests the importance of issuing the query to as many vertical search engines as possible (in order to derive this type of evidence) or caching these post-retrieval features for future impressions of the query. Text-similarity features may have contributed the most to performance because many of the block-types often ranked high in σ_q^* were associated with text-rich information. These included Web blocks w_{1-3}, *news*, *blogs*, and *community Q&A*.

Finally, omitting most feature groups did not result in a significant drop in performance. There are two reasons for this. First, features may be correlated. Second, most verticals are rarely ranked high in σ_q^*. Some of our features may be essential to minority verticals, but this may not have a noticeable effect on the average $K^*(\sigma_q^*, \sigma_q)$. We explore this further in the next section.

8.3 Feature Contribution to Per-Vertical Performance

We also investigated each feature group's contribution to per-vertical ranking performance. Evaluating a feature's contribution to a particular vertical is not trivial. Suppose, for example, that we omit temporal features and want to evaluate the effect on the *news* vertical. One possibility might be to compare the *news* vertical's predicted rank and its ideal rank across a set of queries. However, omitting temporal features may affect *other* verticals as well. And, if so, then ranking mistakes for those verticals would displace *news* from its ideal rank. For this reason, we focus our analysis on the classification approach, which trains one binary classifier per vertical. Each vertical-specific classifier is trained to predict whether the vertical should be displayed (ranked above *eos*) or suppressed (ranked below *eos*) in σ_q.

Performance for a particular vertical is evaluated by considering the quality of confidence values produced by the vertical's corresponding classifier. Let $\mathcal{Q}_v$ denote the queries for which v is a candidate vertical. We evaluate the quality of confidence values output from v's classifier by computing the average precision (AP) metric on a ranking of $\mathcal{Q}_v$ (ranked in descending order of confidence value that v should be presented). In computing AP, a ranked query q is considered "relevant" if $\sigma_q^*(v) < \sigma_q^*(eos)$ and "non-relevant" otherwise. Results in terms of AP are presented in Table 4. We limit our discussion to the case where $p = 5$ because it is associated with verticals ranked higher in σ_q^*.

Table 4 shows several noteworthy trends. First, performance across verticals varied widely (see row "all features"). The best-performing verticals were *weather* (AP=0.938), *finance* (AP=0.638), and *news* (AP=0.594). Interestingly, both *weather* and *finance* were minority verticals. Both were presented (i.e., ranked above *eos* in σ_q^*) for only 14 and 21 queries, respectively. In spite of having few positive instances for training, performance for both these verticals was good. As it turns out, *weather* was easy because *every* query for which it was presented had the query term "weather" (e.g., "weather louisville ky"). This explains why the most predictive feature for *weather* was the click-through feature (i.e., the query's similarity to queries with clicks on weather-related content, many of which contain the term "weather"). Similarly, *finance* was easy because 10/21 queries for which it was presented had the query term "stock(s)" (e.g., "boeing stock"). In other words, performance was high for *weather* and *finance* because their queries often had explicit vertical intent, which is easy to detect.

It is interesting that *news* was among the best performing verticals. This is inconsistent with previous results on vertical selection [2]. The most useful features for *news* were text-similarity features, which are a type of post-retrieval feature. Thus, while it is difficult to detect that a query is newsworthy using only pre-retrieval evidence (as in Arguello *et al.* [2]), useful evidence can be harnessed by issuing the query to the *news* vertical.

The worst performing verticals were *twitter* (AP=0.053), *books* (AP=0.088), and *community Q&A* (AP=0.284). Both *twitter* and *books* were minority verticals, while *community Q&A* was fairly common. Predicting *twitter* may require predicting that the query is about a trending topic, which we did not explicitly address. Predicting *books* and *community Q&A* seems difficult. Queries for which these verticals were relevant had no clear pattern. For *books*, for example, some queries had the keyword "book" (e.g., "books on giraffes"), some corresponded to a book title (e.g., "lolita"), some corresponded to an author name (e.g., "dr. phil"), and, finally, others corresponded to encyclopedic information needs (e.g., "wedding cake ideas", "why don't babies sleep at night", "perennials"). *Community Q&A* queries showed a similar pattern. This may explain why text-similarity features were the only ones to significantly improve performance for both. In the absence of a clearly predictive query-pattern, it seems useful to derive evidence directly from the block.

Features contributed to performance differently for different verticals. Vertical intent features, which exploit vertical-related keywords, were predictive for *maps* and *images*. Many queries for which *maps* was relevant had the keyword "map(s)". Similarly, many queries for which *images* was relevant had the keywords "photo(s)", "pic(s)", and "picture(s)". Category features were predictive for *shopping* because one of our category clusters was related to the shopping ODP node. Different features also hurt performance for different verticals. Named-entity type features hurt performance for *books* and *news*. Category features hurt performance for *community Q&A*, *finance*, *news*, and *recipe*. Click-through features hurt performance for *communtiy Q&A* and *news*.

The only features that did not significantly hurt performance for *any* vertical were text-similarity features. In the previous section, text-similarity features had the greatest contribution to overall performance. In this analysis, text-similarity features were the most consistently predictive for different verticals.

9. CONCLUSION

We proposed and evaluated three general machine learning approaches to block-ranking—ordering blocks of Web and vertical results in response to a query. The best overall performance was obtained by casting this as a learning-to-rank problem. We showed, however, that to be successful, the LTR model must be able to learn a vertical-*specific* relationship between features and block relevance. Our solution to this problem is through feature replication (i.e. by making a block-type-specific copy of each feature). This allows the model to weight a feature as positive evidence for some verticals and negative evidence for others.

Consistent with most prior work in aggregated search, evidence integration is key to block-ranking. Different features were predictive for different verticals. The features that contributed the most to overall performance, and those that were *consistently* predictive for different verticals, were text-similarity features, which are a type of post-retrieval feature. Their contribution to performance suggests the importance of issuing the query to a vertical when possible or of caching this type of evidence for future impressions of the query (or queries similar to it).

10. ACKNOWLEDGMENTS

This work was supported in part by the NSF grant IIS-0916553 and a gift from Yahoo! through its Key Scientific Challenges program. Any opinions, findings, conclusions, and recommendations expressed in this paper are the authors' and do not necessarily reflect those of the sponsors.

11. REFERENCES

[1] J. Arguello, F. Diaz, J. Callan, and B. Carterette. A methodology for evaluating aggregated search results. In *ECIR 2011*, pages 141–152. Springer Berlin / Heidelberg, 2011.

[2] J. Arguello, F. Diaz, J. Callan, and J.-F. Crespo. Sources of evidence for vertical selection. In *SIGIR 2009*, pages 315–322. ACM, 2009.

[3] J. Arguello, F. Diaz, and J.-F. Paiement. Vertical selection in the presence of unlabeled verticals. In *SIGIR 2010*, pages 691–698. ACM, 2010.

[4] D. M. Bikel, R. Schwartz, and R. M. Weischedel. An algorithm that learns what is in a name. *Machine Learning*, 34:211–231, 1999.

[5] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees.* Chapman and Hall/CRC, 1984.

[6] F. Diaz. Integration of news content into web results. In *WSDM 2009*, pages 182–191. ACM, 2009.

[7] F. Diaz and J. Arguello. Adaptation of offline vertical selection predictions in the presence of user feedback. In *SIGIR 2009*, pages 323–330. ACM, 2009.

[8] H. Jeffreys. An invariant form for the prior probability in estimation problems. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 186(1007):453–461, 1946.

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

[10] A. C. König, M. Gamon, and Q. Wu. Click-through prediction for news queries. In *SIGIR 2009*, pages 347–354. ACM, 2009.

[11] R. Kumar and S. Vassilvitskii. Generalized distances between rankings. In *WWW 2010*, pages 571–580. ACM, 2010.

[12] X. Li, Y.-Y. Wang, and A. Acero. Learning query intent from regularized click graphs. In *SIGIR 2008*, pages 339–346. ACM, 2008.

[13] C.-J. Lin, R. C. Weng, and S. S. Keerthi. Trust region newton methods for large-scale logistic regression. In *ICML 2007*, pages 561–568. ACM, 2007.

[14] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Informaton Retrieval*, 3:225–331, 2009.

[15] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval.* Cambridge University Press, New York, NY, USA, 2008.

[16] A. K. Ponnuswami, K. Pattabiraman, Q. Wu, R. Gilad-Bachrach, and T. Kanungo. On composition of a federated web search result page: Using online users to provide pairwise preference for heterogeneous verticals. In *WSDM 2011*, pages 715–724. ACM, 2011.

[17] M. Schulze. A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method. *Social Choice and Welfare*, 36:267–303, 2011.